Introduction to Requirements Engineering

1. Definition

Requirements Engineering (RE) is the systematic process of discovering, analyzing, documenting, validating, and managing the requirements for a software system.

It ensures that the final software product meets the actual needs of stakeholders while being technically feasible, cost-effective, and reliable.

In simple words, it is the process of making sure we build the right system by understanding what the users want and ensuring developers implement it correctly.

> 💡 **Example:** A hospital wants a software system to manage patient records. Without RE, developers may create a system that records names and ages but ignores sensitive details like medical history or insurance information. With RE, all these details are carefully captured before development begins.

2. Importance of Requirements Engineering

Requirements Engineering is considered the foundation of Software Engineering, because all later activities (design, coding, testing, maintenance) depend on it.

Avoids Miscommunication

It provides a clear channel of communication between stakeholders and developers, ensuring everyone has the same understanding.

Saves Time and Cost

Errors found in requirements are cheaper to fix than errors found during coding or after release.

Example: If customers expect online payment in an app but developers only build cash-on-delivery, fixing it after release is very costly.

Created with formatmagic.ai - One click automated formatting

Defines Project Scope

It clearly states what the system will do and what it will not do, helping avoid scope creep (uncontrolled expansion of features).

Supports Testing and Validation

Requirements act as a reference for testers to confirm whether the system works as intended.

Ensures Customer Satisfaction

By capturing actual user needs, RE helps deliver software that truly satisfies users.

3. Objectives of Requirements Engineering

The main goals of RE are:

- **To Identify Needs:** Discover what stakeholders really want from the system.
- **To Analyze and Negotiate:** Resolve conflicts among stakeholders with different priorities.
- **To Specify Requirements:** Document requirements in a clear and structured form.
- **To Validate Requirements:** Ensure that documented requirements are correct, complete, and realistic.
- **To Manage Requirements:** Keep track of changes as the project evolves.

4. The Requirements Engineering Process

The RE process involves several important activities:

Requirements Elicitation

Gathering requirements from users, customers, and domain experts.

Techniques include interviews, questionnaires, workshops, and observations.

Example: For a school system, interviewing teachers to understand how they want attendance to be recorded.

Created with formatmagic.ai - One click automated formatting

Requirements Analysis

Studying collected requirements to detect conflicts, missing parts, and feasibility issues.

Helps prioritize important features.

Example: Parents may want a mobile app for student progress, but the school may only afford a website. Analysis helps in decision-making.

Requirements Specification

Documenting requirements in a Software Requirements Specification (SRS) document.

Ensures clarity, avoids ambiguity, and acts as a contract between client and developer.

Example: Instead of saying "system should be fast," specify: "System shall load dashboard within 3 seconds for 90% of users."

Requirements Validation

Checking requirements to ensure they are correct and meet user expectations.

Techniques: reviews, prototypes, test case generation.

Example: Showing a prototype of an e-commerce checkout system to the client for confirmation.

Requirements Management

Handling changes in requirements during the software lifecycle.

Includes tracking versions, updating documents, and communicating changes.

Example: A new tax law may require updating an accounting system to calculate additional VAT.

5. Types of Requirements in RE

- Functional Requirements – Define what the system should do.

Created with formatmagic.ai - One click automated formatting

Example: “System shall allow customers to track orders.”

- Non-Functional Requirements – Define how the system should perform.

Example: “System must support 1000 users at the same time.”

- Domain Requirements – Industry or field-specific requirements.

Example: “Hospital system must comply with patient privacy laws (HIPAA).”

6. Challenges in Requirements Engineering

- **Ambiguity:** Stakeholders use vague terms like “system should be easy to use.”
- **Changing Requirements:** Business needs change during development.
- **Conflicting Interests:** Different stakeholders demand contradictory features.
- **Communication Gap:** Technical developers may not fully understand business terms.
- **Incomplete Knowledge:** Sometimes stakeholders don’t know what they want until they see a prototype.

7. Benefits of Good Requirements Engineering

- Reduces project risks and chances of failure.
- Produces higher-quality software.
- Increases customer satisfaction.
- Saves time and cost by reducing rework.
- Provides a strong foundation for design, coding, and testing.

✅ Summary (Quick Exam Recall):

- Requirements Engineering = Process of eliciting, analyzing, specifying, validating, and managing software requirements.
- Ensures software meets user needs, is feasible, and is cost-effective.
- **Types of requirements:** Functional, Non-Functional, and Domain.
- **Activities:** Elicitation → Analysis → Specification → Validation → Management.
- **Benefits:** Better communication, reduced cost, high quality, satisfied users.

Created with formatmagic.ai - One click automated formatting

Software Requirements

1. Definition

Software Requirements are the statements that describe what a software system should do and how it should behave under specific conditions. They act as a blueprint for the software development process and bridge the gap between the user's needs and the developer's implementation.

In simple terms:

- They define what services the system should provide (functional aspects).
- They also define the quality and constraints under which the system should operate (non-functional aspects).

Without well-defined software requirements, the development team risks building the wrong product, wasting time, money, and effort.

💡 **Example:** For an e-commerce website, a requirement could be:

Functional: "The system shall allow customers to add products to the cart."

Non-Functional: "The cart must update within 1 second after an item is added."

2. Importance of Software Requirements

Software requirements are considered the foundation of Software Engineering. Their importance can be explained as follows:

- **Clarity Between Stakeholders and Developers**
  Requirements serve as a communication tool between customers and the development team. They ensure that both parties have the same understanding of the software to be built.
  Example: If the requirement is simply "the system should support payments," the customer may expect multiple payment gateways, but the developer may only implement credit card payments. Clear requirements prevent this mismatch.

Created with formatmagic.ai - One click automated formatting

- **Basis for Design and Development**
  Requirements define the scope of the software. Designers and developers use them as a guide to create system architecture and write code. Without them, design may lack direction.
- **Testing and Validation**
  Testers rely on requirements to prepare test cases. A software product can only be validated if there is a written requirement to compare it against.
- **Project Planning**
  Time, cost, manpower, and resources are estimated based on the scope of requirements. Incorrect or incomplete requirements lead to incorrect estimations.
- **Minimizing Risk of Failure**
  Studies show that many software projects fail due to poor requirement engineering. Clearly defined requirements reduce the chances of delivering software that does not meet user needs.

3. Characteristics of Good Software Requirements

For software requirements to be useful, they must have certain qualities. Each characteristic is explained below:

- **Correct**
  The requirement should represent the actual needs of the stakeholders. If requirements do not reflect what users want, the software will not solve their problems.
  Example: In a hospital management system, if doctors need a "prescription printing" feature but it is not included in requirements, the system will be considered incomplete.
- **Complete**
  A requirement must cover all possible inputs, outputs, and scenarios. An incomplete requirement may leave gaps in the system.
  Example: Login functionality should not only state "system shall allow login" but also include error handling like wrong password, account lock, and forgot password options.
- **Consistent**
  Requirements should not contradict each other. If two requirements conflict, developers will be confused.
  Example: Requirement 1 says "System must store user passwords using encryption," while Requirement 2 says "Passwords should be retrievable in plain text." These are inconsistent.

Created with formatmagic.ai - One click automated formatting

- **Unambiguous**
  A requirement should have only one interpretation. Ambiguity creates misunderstandings.
  Example: Instead of saying "System should be fast," write "System should load dashboard within 2 seconds for 95% of users."
- **Feasible**
  A requirement should be technically possible to implement within time, cost, and resource limits.
  Example: "The system should process 1 million transactions per second on a small desktop machine" is unrealistic.
- **Verifiable**
  A requirement must be testable. There should be a way to confirm whether it has been met or not.
  Example: "System should provide good performance" is not verifiable, but "System should respond within 3 seconds under 100 concurrent users" is.
- **Prioritized**
  Not all requirements have equal importance. Some are critical, while others are optional. Prioritization helps in case of limited time or budget.

4. Types of Software Requirements

A. Functional Requirements (FRs)

Functional requirements specify the services, tasks, and functions the system must perform. They describe how the system should react to specific inputs and behave in different situations.

Examples:

- The system shall allow users to register and log in.
- The system shall generate a monthly report of sales.
- The ATM shall dispense cash when the correct PIN is entered.

👉 These are essential because they define the core functionality of the system.

B. Non-Functional Requirements (NFRs)

Non-functional requirements describe the quality attributes of a system rather than its behavior. They are sometimes called constraints or quality requirements.

Created with formatmagic.ai - One click automated formatting

Categories of NFRs:

- **Performance:** Defines speed and efficiency.
  Example: The website must handle 5000 users simultaneously.
- **Security:** Defines protection against threats.
  Example: All passwords must be encrypted using SHA-256.
- **Usability:** Defines user-friendliness.
  Example: The system must provide tooltips for all buttons.
- **Reliability:** Defines system stability.
  Example: System uptime must be at least 99.9%.
- **Portability:** Defines adaptability.
  Example: The app must run on Android, iOS, and Windows.

👉 These ensure that the system is not only functional but also usable, reliable, and efficient.

C. Domain Requirements

These are requirements that arise from the specific domain or industry in which the software will operate. They often come from legal, business, or operational rules.

Examples:

- A banking system must comply with PCI-DSS standards for secure card transactions.
- A hospital management system must follow HIPAA regulations for patient privacy.

👉 Domain requirements are critical because ignoring them can make the software legally or operationally unusable.

5. Sources of Software Requirements

Requirements are gathered from different sources. The main ones are:

- **Stakeholders:** Users, customers, managers who directly state their needs.
- **Business Documents:** Policies, contracts, and process manuals that define system rules.
- **Existing Systems:** Old software provides insight into missing or improved features.
- **Domain Experts:** Specialists who know the rules and standards of the industry.

Created with formatmagic.ai - One click automated formatting

- **Prototypes and Models:** Mockups or early versions that reveal hidden requirements.

6. Problems in Defining Software Requirements

- **Ambiguity:** Natural language statements may be unclear.
- **Changing Requirements:** User needs evolve due to business or market shifts.
- **Conflicts Among Stakeholders:** Different users may want different features.
- **Incomplete Coverage:** Some scenarios may be overlooked.
- **Over-Specification:** Adding too much detail may restrict developer creativity.

7. Role of Software Requirements in Software Development

- Act as a contract between client and developers.
- Provide the scope and boundaries of the project.
- Serve as the foundation for design, coding, and testing.
- Used in project estimation of cost, time, and resources.
- Help in validating final software against expectations.

✅ **Summary (Quick Recall for Exams):**

- Software Requirements = Functional + Non-Functional + Domain needs.
- They must be correct, complete, consistent, unambiguous, feasible, verifiable, and prioritized.
- Functional → What the system does.
- Non-Functional → How the system behaves (performance, security, usability, etc.).
- Domain → Industry-specific needs.
- Requirements come from stakeholders, documents, experts, old systems, and prototypes.
- They guide design, coding, testing, and validation of the software.

Created with formatmagic.ai - One click automated formatting

Classification of Requirements

1. Introduction

Requirements can be classified into different categories to better understand, organize, and manage them. This classification helps developers, testers, and stakeholders to clearly differentiate between what the system must do, how it should perform, and what external rules or constraints it must follow.

Without classification, all requirements may look the same, leading to confusion, poor prioritization, and increased project risk.

2. Main Classification of Requirements

The most widely accepted classification divides requirements into three main categories:

- Functional Requirements (FRs)
- Non-Functional Requirements (NFRs)
- Domain Requirements

Let's explain each in detail with examples.

A. Functional Requirements (FRs)

Definition:
Functional requirements describe the specific functions, tasks, and services the system must provide. They are concerned with what the system should do.

Explanation:
They define the interaction between the user and the system, including inputs, outputs, and behaviors. These requirements form the core of the system functionality.

Examples:

- **In a banking system:** "The system shall allow users to transfer funds between accounts."
- **In a shopping website:** "The system shall allow customers to add products to a shopping cart."

Created with formatmagic.ai - One click automated formatting

- **In an ATM:** “The ATM shall dispense cash when the correct PIN is entered.”

👉 Why important? Because they determine the primary purpose of the software. If functional requirements are incomplete or wrong, the system will not meet user needs.

B. Non-Functional Requirements (NFRs)

Definition:
Non-functional requirements specify the quality attributes, performance standards, and constraints of a system. They define how the system should behave, rather than what it does.

Explanation:
These requirements affect the user experience, system performance, security, and maintainability. Even if all functional requirements are correct, a system may still fail if non-functional requirements are ignored.

Categories & Examples:

- **Performance:**
  - “The system shall support 10,000 concurrent users.”
- **Usability:**
  - “The system must provide a help guide and tooltips for all features.”
- **Reliability:**
  - “The system must have 99.9% uptime per month.”
- **Security:**
  - “All passwords must be encrypted using SHA-256 algorithm.”
- **Portability:**
  - “The system should run on Windows, Linux, and Mac OS.”
- **Maintainability:**
  - “The system should allow adding new modules without affecting existing ones.”

👉 Why important? Because they determine the quality and success of the software in real-world usage.

Created with formatmagic.ai - One click automated formatting

C. Domain Requirements

Definition:
Domain requirements are industry-specific rules, laws, or standards that the software must follow. They originate from the environment in which the software will be used.

Explanation:
They often come from government regulations, business rules, or safety standards. These requirements are unique to each industry or domain.

Examples:

- **In healthcare:** "The system must comply with HIPAA regulations for patient privacy."
- **In finance:** "The system must follow PCI-DSS standards for secure credit card transactions."
- **In aviation:** "The software must comply with FAA safety standards."

👉 Why important? Because ignoring them can make the software illegal, unsafe, or unusable in its intended environment.

3. Additional Classifications

- **User Requirements**
  - Written in natural language for end-users.
  - **Example:** "The user should be able to reset their password easily."
- **System Requirements**
  - More detailed, technical requirements for developers.
  - **Example:** "System shall provide password reset functionality using OTP verification."
- **Business Requirements**
  - High-level goals of the organization.
  - **Example:** "The system should increase online sales by 30% in one year."
- **Regulatory Requirements**
  - Requirements imposed by government or legal authorities.
  - **Example:** "System must retain tax data for 7 years as per law."

Created with formatmagic.ai - One click automated formatting

4. Differences Between FR, NFR, and Domain Requirements

ASPECT	FUNCTIONAL REQUIREMENTS	NON-FUNCTIONAL REQUIREMENTS	DOMAIN REQUIREMENTS
FOCUS	What the system does	How the system performs	Industry-specific rules
NATURE	Services and tasks	Quality attributes & constraints	Laws, regulations, business rules
EXAMPLE (E-COMMERCE)	"System shall allow order placement"	"System shall load pages in < 2 sec"	"System must follow online tax rules"
IMPORTANCE	Defines core functionality	Ensures usability & performance	Ensures legal/ operational compliance

5. Challenges in Requirement Classification

- Some requirements may overlap categories (e.g., security can be functional or non-functional).
- Stakeholders often describe requirements in natural language, making classification difficult.
- Misclassification may lead to incomplete testing or missing quality aspects.

6. Summary (Quick Recall)

- Classification of Requirements helps in better understanding and management of software needs.
- **Functional Requirements (FRs):** What the system should do (core services).
- **Non-Functional Requirements (NFRs):** How the system should perform (quality attributes).
- **Domain Requirements:** Industry-specific rules or laws.
- **Additional types:** User, System, Business, and Regulatory Requirements.
- Correct classification ensures clarity, completeness, and successful system development.

Created with formatmagic.ai - One click automated formatting

Requirements Process

1. Introduction

The Requirements Process is the structured set of activities used to discover, analyze, document, validate, and manage the requirements of a software system. It is the heart of Requirements Engineering and ensures that the final system truly reflects what stakeholders need.

Without a proper requirements process, software projects may face:

- Misunderstood user needs
- Cost overruns
- Missed deadlines
- System failure

> 💡 **Example:** If a mobile banking app is developed without a proper requirements process, developers may miss critical needs like two-factor authentication, leading to customer dissatisfaction and security risks.

2. Importance of the Requirements Process

- Provides clarity on what the system must deliver.
- Reduces miscommunication between users and developers.
- Helps in early detection of errors.
- Acts as a foundation for design, development, and testing.
- Makes the project more manageable through documentation and change control.

3. Steps in the Requirements Process

The requirements process typically consists of the following five main activities:

Created with formatmagic.ai - One click automated formatting

1. Requirements Elicitation

Definition: The process of gathering requirements from stakeholders, domain experts, documents, and existing systems.

Explanation: Since users often cannot express everything clearly, elicitation involves asking questions, observing, and conducting workshops to uncover both stated and hidden needs.

Techniques Used:

- Interviews
- Questionnaires
- Brainstorming sessions
- Observation of current processes
- Prototyping

Example: In a hospital system, interviewing doctors, nurses, and administrative staff to collect requirements about patient registration, billing, and treatment records.

2. Requirements Analysis

Definition: The process of examining gathered requirements to detect problems like conflicts, ambiguities, infeasibility, or duplication.

Explanation: Analysis ensures that requirements are well understood, consistent, and prioritized.

Activities in Analysis:

- Detect conflicts among stakeholders.
- Prioritize requirements (critical vs. optional).
- Check feasibility within time, budget, and technology constraints.

Example: In a shopping website, one group of stakeholders wants a desktop-only platform while others demand a mobile app. Analysis resolves such conflicts by weighing business needs.

3. Requirements Specification

Definition: The process of documenting the requirements in a structured, unambiguous, and precise manner.

Created with formatmagic.ai - One click automated formatting

Explanation: The outcome of this step is the Software Requirements Specification (SRS) document, which serves as a contract between client and developer.

Contents of SRS typically include:

- Functional requirements
- Non-functional requirements
- Domain requirements
- Constraints and assumptions

Example: Instead of writing "system should be secure," the specification states: "System shall encrypt all passwords using AES-256 encryption."

4. Requirements Validation

Definition: The process of ensuring that documented requirements are correct, complete, and consistent with stakeholder needs.

Explanation: Validation ensures that the right system is being built before actual coding begins.

Techniques Used:

- Requirement reviews
- Prototyping and user feedback
- Model validation
- Test case generation

Example: Showing a prototype of an e-commerce checkout page to stakeholders to confirm if the flow matches their expectations.

5. Requirements Management

Definition: The continuous process of handling changes to requirements as the project progresses.

Explanation: Since business needs, market trends, and technology keep changing, requirements must be tracked, versioned, and updated.

Activities in Management:

- Tracking requirement changes
- Version control of documents
- Traceability (linking each requirement to design, code, and test cases)

Created with formatmagic.ai - One click automated formatting

- Impact analysis of changes

Example: A government introduces a new sales tax. The e-commerce system must be updated to calculate this tax. Requirements management ensures smooth incorporation of such changes.

4. Challenges in the Requirements Process

- Stakeholders may not know what they want.
- Conflicting requirements from different users.
- Constantly changing business environment.
- Ambiguity in natural language requirements.
- Difficulty in prioritizing requirements when all seem important.

5. Benefits of a Well-Defined Requirements Process

- Improved communication between stakeholders and developers.
- Fewer errors and reduced cost of fixing mistakes.
- Higher customer satisfaction.
- Clear documentation for future reference.
- Better project control and predictability.

6. Requirements Process Flow (Diagram in Text Form)

Requirements Elicitation → Requirements Analysis → Requirements Specification → Requirements Validation → Requirements Management

This flow is iterative, meaning steps may repeat until requirements are stable and agreed upon.

Created with formatmagic.ai - One click automated formatting

✅ Summary (Quick Exam Recall)

- Requirements Process = Structured activities to gather, analyze, specify, validate, and manage requirements.
- **Steps:** Elicitation → Analysis → Specification → Validation → Management.
- Ensures software is correct, complete, feasible, and aligned with user needs.
- **Benefits:** Better communication, reduced risk, cost savings, and improved quality.

Created with formatmagic.ai - One click automated formatting

Levels/Layers of Requirements

In Software Requirement Engineering, requirements are not all of the same type or detail. To make them manageable and understandable, they are organized into different levels (or layers). Each level describes requirements from a different perspective, starting from the business needs down to the technical details.

This layered approach ensures that:

- The business goals are clear.
- The user's needs are understood.
- The system's functionality is properly defined.
- The technical specifications are precise for developers.

1. Business Requirements (High-Level Requirements)

Definition: These describe the overall objectives, vision, and goals of the organization or project. They explain why the system is needed.

Purpose: To capture the business value and strategic needs.

Example:

A bank wants to increase customer satisfaction by providing an online banking system.

Business requirement → "Develop an online system that allows customers to access their accounts 24/7."

2. User Requirements (Stakeholder Requirements)

Definition: These describe what the end-users or stakeholders expect from the system in their own language (not technical).

Purpose: To capture user goals, tasks, and expectations.

Form: Usually written as natural language statements, use cases, or user stories.

Example:

- "The user should be able to transfer money from one account to another."
- "The user should be able to view the balance of their account in real-time."

Created with formatmagic.ai - One click automated formatting

3. System Requirements

Definition: These describe in detail what the system should do to fulfill user requirements. They are more precise and technical.

Types:

Functional Requirements → Describe the functions/features of the system.

Example: "The system shall generate a monthly account statement in PDF format."

Non-Functional Requirements → Describe the quality attributes (performance, security, usability, reliability).

Example: "The system should support up to 10,000 concurrent users with a response time of less than 2 seconds."

4. Design/Technical Requirements

Definition: These specify the technical details and constraints required for implementation. They are meant for developers, architects, and testers.

Purpose: To translate system requirements into design specifications.

Examples:

- "The system shall be developed using Java Spring Boot and MySQL."
- "The mobile application must run on Android 10 and above."

5. Transition Requirements (Optional Layer)

Definition: These specify the temporary conditions and activities needed to move from the old system to the new system.

Examples:

- "Data from the old banking system must be migrated to the new system before deployment."
- "Staff should be trained to use the new system within 2 weeks of launch."

Diagram (Levels of Requirements)

Business Requirements

↓

User Requirements

Created with formatmagic.ai - One click automated formatting

↓

System Requirements

↓

Design/Technical Requirements

↓

Transition Requirements

Summary

Business Requirements → Why the system is needed (organization goals).

User Requirements → What users expect (user tasks & needs).

System Requirements → What the system must do (functional + non-functional).

Design/Technical Requirements → How the system will be built (platforms, technologies).

Transition Requirements → Temporary needs for migration & adoption.

✅ This layered approach ensures clarity, traceability, and completeness of requirements.

Created with formatmagic.ai - One click automated formatting

Requirement Characteristics

In Software Engineering, not all requirements are automatically good. A requirement must have certain qualities (characteristics) to be clear, useful, and testable. These characteristics help ensure that requirements are well-defined, complete, and implementable.

If requirements lack these qualities, the system may face problems like misunderstandings, errors, cost overruns, or project failure.

Key Characteristics of Good Requirements

1. Correct

Definition: The requirement must accurately describe what the system should do, without mistakes.

Why Important: Wrong requirements lead to wrong system functionality.

Example:

Correct → "The system shall allow users to reset their password using email verification."

Incorrect → "The system shall allow users to reset their password without any verification." (security risk).

2. Complete

Definition: The requirement must contain all necessary details without leaving gaps.

Why Important: Incomplete requirements cause ambiguity and missing features.

Example:

Incomplete → "The system shall generate reports." (What kind of reports? Format?)

Complete → "The system shall generate monthly sales reports in PDF format including product name, quantity, and revenue."

Created with formatmagic.ai - One click automated formatting

3. Clear/Unambiguous

Definition: The requirement must be stated in a single clear meaning, not open to multiple interpretations.

Why Important: Ambiguity confuses developers and testers.

Example:

Ambiguous → "The system should be fast." (Fast means 1 sec? 5 sec?)

Clear → "The system shall load the dashboard within 3 seconds for 95% of requests."

4. Consistent

Definition: The requirement must not contradict other requirements.

Why Important: Conflicting requirements create confusion and wasted effort.

Example:

Inconsistent:

Req 1: "The system shall send password reset via SMS."

Req 2: "The system shall not use SMS services."

Consistent: Both should align (either allow SMS or not).

5. Feasible (Realistic)

Definition: The requirement must be possible to implement with available resources, technology, time, and budget.

Why Important: Unrealistic requirements cause project failure.

Example:

Unrealistic → "The system shall process 10 million transactions per second on a single PC."

Feasible → "The system shall process 10,000 transactions per second on a distributed server setup."

Created with formatmagic.ai - One click automated formatting

6. Verifiable (Testable)

Definition: The requirement must be measurable and testable through inspection, demonstration, or testing.

Why Important: If it cannot be tested, we cannot prove it works.

Example:

Non-testable → "The system should be user-friendly."

Testable → "The system shall allow a new user to complete registration within 3 minutes without training."

7. Traceable

Definition: Each requirement should be linked to its source (business goal, user need, regulation, etc.) and traceable through design, development, and testing.

Why Important: Helps in tracking changes and ensuring nothing is missed.

Example:

Business Goal → "Reduce checkout time."

User Requirement → "One-click payment option."

System Requirement → "Integrate payment gateway with saved card feature."

8. Prioritized

Definition: Requirements should be ranked based on importance and urgency.

Why Important: Not all requirements can be implemented at once. Prioritization helps in managing scope.

Example:

High Priority → "The system must provide secure login with multi-factor authentication."

Low Priority → "The system should allow custom dashboard themes."

9. Modifiable

Definition: The requirement should be written in a way that it can be changed easily when necessary, without affecting unrelated requirements.

Created with formatmagic.ai - One click automated formatting

Why Important: Projects evolve, so requirements must adapt.

Example:

Poor → Requirements written in long paragraphs mixing multiple needs.

Good → Each requirement is numbered and written separately.

10. Understandable

Definition: Requirements should be written in simple, clear language so that all stakeholders (business, users, developers, testers) can understand.

Why Important: Misunderstood requirements cause errors in implementation.

Example:

Bad → Technical jargon-heavy statements.

Good → User-focused, plain language.

Summary Table

CHARACTERISTIC	MEANING	EXAMPLE (GOOD VS BAD)
CORRECT	Must be accurate	✅ Email verification reset / ❌ No verification
COMPLETE	Covers all details	✅ PDF monthly report / ❌ Just “generate report”
CLEAR	No ambiguity	✅ 3 sec load / ❌ “Fast”
CONSISTENT	No contradictions	✅ SMS allowed / ❌ SMS forbidden + required
FEASIBLE	Realistic to implement	✅ 10k/sec / ❌ 10M/sec on 1 PC

Created with formatmagic.ai - One click automated formatting

CHARACTERISTIC	MEANING	EXAMPLE (GOOD VS BAD)
VERIFIABLE	Can be tested	✅ Registration ≤ 3 min / ❌ “User-friendly”
TRACEABLE	Linked to source	✅ Goal → Req → System
PRIORITIZED	Ranked importance	✅ MFA login (high) / ❌ Dashboard theme (low)
MODIFIABLE	Easy to update	✅ Numbered, separated
UNDERSTANDABLE	Easy to read	✅ Plain English / ❌ Complex jargon

Created with formatmagic.ai - One click automated formatting

Analyzing Quality Requirements

1. Introduction

Quality requirements (also called non-functional requirements, NFRs) describe how well the system should perform rather than what functions it performs.

- **Examples:** performance, reliability, usability, maintainability, portability, security.

Analyzing quality requirements ensures the system is efficient, secure, reliable, and user-friendly.

2. Why Analyze Quality Requirements?

- To avoid vague requirements like “System should be fast” → not measurable.
- To make sure requirements are testable, clear, and realistic.
- To balance conflicting qualities (e.g., high security vs. usability).
- To prioritize qualities based on stakeholder needs.

3. Steps in Analyzing Quality Requirements

Step 1: Identify Quality Attributes

Collect all required qualities from stakeholders, standards, and domain knowledge.

Common attributes:

- **Performance:** Response time, throughput.
- **Reliability:** Fault tolerance, recovery.
- **Usability:** Easy to learn, user satisfaction.
- **Security:** Protection against attacks.
- **Portability:** Works on different platforms.

Example:

- Banking app → requires high security, high availability, but also good usability for customers.

Created with formatmagic.ai - One click automated formatting

Step 2: Quantify Requirements

Replace vague terms with measurable metrics.

Techniques:

- Use numbers, percentages, or time limits.
- Define acceptance criteria.

Examples:

- Instead of “System should be fast” → “System must process 95% of transactions within 2 seconds.”
- Instead of “System should be reliable” → “System uptime must be 99.9% per year.”

Step 3: Prioritize Requirements

Not all qualities can be maximized together (e.g., security vs. usability).

Prioritize using:

- **MoSCoW method:** Must have, Should have, Could have, Won’t have.
- **Trade-off analysis:** Balance cost vs. benefit.

Example:

- For a military system, security > usability.
- For a mobile gaming app, usability & performance > strict security.

Step 4: Analyze Conflicts

Some requirements contradict each other.

Examples:

- Security vs. Usability → More password checks = harder for users.
- Performance vs. Maintainability → Optimized code may be harder to maintain.

Solution:

- Negotiate with stakeholders.
- Apply architectural trade-offs (e.g., caching improves performance but may affect freshness of data).

Created with formatmagic.ai - One click automated formatting

Step 5: Validation of Quality Requirements

Check if requirements are:

- **Complete:** All qualities included.
- **Consistent:** No contradictions.
- **Measurable:** Clear metrics defined.
- **Testable:** Can be verified during testing.

4. Tools & Techniques for Analysis

Quality Attribute Scenarios (QAS): Structured format for describing NFRs.

Example for Performance:

- **Stimulus:** User requests a report.
- **Response:** System generates the report in < 5 seconds.

- **Checklists:** Ensure all common NFRs are considered.
- **Prototyping:** Test usability, performance early.
- **Modeling & Simulation:** Test reliability, load handling before deployment.

5. Example Case

E-commerce Website

Identified quality requirements:

- Performance → Handle 1000 users simultaneously with < 3 sec response time.
- Security → All transactions must use 256-bit encryption.
- Reliability → System uptime 99.95%.
- Usability → Average new user should complete checkout within 5 minutes.

Analysis showed: High security slowed checkout → solution: one-click checkout with encryption.

6. Conclusion

Analyzing quality requirements is crucial for building practical, efficient, and user-accepted systems.

Created with formatmagic.ai - One click automated formatting

It involves identifying, quantifying, prioritizing, resolving conflicts, and validating non-functional requirements.

Good analysis prevents vague, conflicting, or unrealistic quality demands.

Created with formatmagic.ai - One click automated formatting

Software Requirements in the Context of Systems Engineering

1. Introduction

Software does not exist in isolation — it is usually part of a larger system (e.g., a hospital system, banking system, traffic control system, or even an aircraft system).

Systems Engineering (SE) is the broader discipline that deals with designing, analyzing, and managing entire systems, while Software Requirement Engineering (SRE) focuses specifically on the software part of that system.

Thus, software requirements must be developed within the larger framework of systems engineering.

2. Relationship between Systems Engineering and Software Requirements

Systems Engineering defines the overall objectives of the system, considering hardware, people, environment, and processes.

Software Requirements Engineering ensures that the software component meets those overall objectives.

Example:

A Banking System may include ATM machines, networks, security policies, and staff.

The software requirements would define the ATM software behavior (e.g., PIN validation, balance checking, fund transfer).

3. Why Software Requirements Depend on Systems Engineering

Context Awareness

Software must be aware of the environment it works in (e.g., sensors, hardware, networks).

Created with formatmagic.ai - One click automated formatting

Example: In an aircraft, the autopilot software must integrate with sensors and physical control mechanisms.

Constraints

System-level constraints (cost, hardware, safety, regulatory rules) shape software requirements.

Example: A medical device system must follow FDA safety regulations, so the software must include audit trails.

Interfaces

Software interacts with other system components. Requirements must define these interfaces clearly.

Example: A traffic control system's software must interface with hardware like cameras and signals.

4. Hierarchical View

Systems Engineering typically has multiple layers:

- System Requirements → Define what the overall system should achieve.
- Software Requirements → Derived from system requirements, focusing on the software's role.
- Detailed Design & Implementation → Translate requirements into code and architecture.

🔹 **Example:** In a Hospital Management System:

- **System Requirement:** The system should manage patient records securely.
- **Software Requirement:** The software must provide encrypted database storage and role-based access for doctors/nurses.

5. Importance of Software Requirements in Systems Engineering

- Alignment → Ensures software supports system-level goals.
- Risk Reduction → Identifies technical, safety, or performance risks early.
- Traceability → Requirements can be traced back to higher-level system needs.

Created with formatmagic.ai - One click automated formatting

- Stakeholder Satisfaction → Connects user/system needs with software functions.

6. Examples from Real Systems

Air Traffic Control System

- **System Requirement:** Ensure safe flight operations.
- **Software Requirement:** The radar software must track aircraft positions in real time with 1-second updates.

E-commerce System

- **System Requirement:** Allow customers to purchase goods online.
- **Software Requirement:** The website must support shopping cart functionality, secure payment, and order tracking.

7. Challenges

- Complexity → Software must fit within a large, complex system.
- Communication Gap → System engineers may focus on hardware, while software engineers focus on code, causing misunderstandings.
- Change Management → Any change in system requirements may ripple down to software requirements.
- Verification → Ensuring that software requirements meet system-level requirements is often difficult.

8. Summary (Exam-Ready Points)

- Software requirements must be understood in the larger system context.
- They are derived from system requirements and must align with system goals.
- They ensure correct functionality, performance, safety, and compliance of the system.
- Good communication between system engineers and software engineers is essential.
- **Example domains:** banking systems, healthcare, aviation, defense, transportation.

Created with formatmagic.ai - One click automated formatting

Requirement Evolution

1. Introduction

Requirements are not static — they change over time due to various internal and external factors. This process of continuous change, refinement, and growth of requirements throughout the software/system lifecycle is called Requirement Evolution.

It is a natural phenomenon in software engineering because software must adapt to new business needs, technology updates, regulations, and user expectations.

2. Definition

Requirement Evolution is the process of managing and adapting software requirements as they change over the system's lifecycle due to new needs, environmental changes, or stakeholder feedback.

3. Why Do Requirements Evolve? (Causes)

Changing Business Environment

Market competition forces new features.

Example: E-commerce apps adding digital wallets due to competition.

Stakeholder Needs

Users discover new requirements after using the system.

Example: Students request an online portal to download assignments.

Technological Advances

New platforms or hardware require changes.

Example: Mobile-first approach requires modifying web-based requirements.

Regulatory & Legal Changes

Laws and compliance requirements evolve.

Created with formatmagic.ai - One click automated formatting

Example: GDPR laws forced changes in data privacy requirements.

Error in Original Requirements

Misunderstandings or incomplete requirements need correction.

Organizational Changes

Mergers, acquisitions, or strategy shifts impact system goals.

4. Aspects of Requirement Evolution

- Addition of Requirements – New features or services are introduced.
- Modification of Requirements – Existing requirements are changed to improve accuracy.
- Deletion of Requirements – Some requirements may be removed due to redundancy or cost.
- Prioritization – Requirements may be re-ordered depending on business value.

5. Process of Requirement Evolution

Requirement evolution is handled using Requirement Management practices, including:

- Identification
- Analysis & Impact Assessment
- Approval & Negotiation
- Implementation & Documentation
- Traceability & Verification

6. Challenges in Requirement Evolution

- Cost & Time Overruns – Frequent changes delay projects.
- Scope Creep – Uncontrolled addition of features.
- Complexity – Changes may conflict with existing requirements.
- Communication Gaps – Misalignment between stakeholders and developers.
- Version Control – Difficulty in managing requirement documents.

Created with formatmagic.ai - One click automated formatting

7. Example Scenarios

Banking System

Original Requirement: Provide ATM cash withdrawal.

Evolved Requirement: Add biometric authentication for security.

Healthcare System

Original Requirement: Store patient records.

Evolved Requirement: Enable cloud access for doctors working remotely.

Social Media App

Original Requirement: Post text and images.

Evolved Requirement: Support video stories and reels (due to user demand & competition).

8. Importance of Requirement Evolution

- Ensures software remains relevant to changing needs.
- Improves user satisfaction by adapting to feedback.
- Enhances competitiveness in the market.
- Provides regulatory compliance.
- Protects investment in long-term projects by allowing adaptation.

9. Summary (Exam-Ready Points)

- Requirement Evolution = continuous change in requirements during the software lifecycle.
- Driven by business, users, technology, regulations, and errors in original requirements.
- Involves addition, modification, deletion, and prioritization.
- Managed through requirement management & traceability.
- Essential for keeping software relevant, compliant, and competitive.

Created with formatmagic.ai - One click automated formatting

Requirement Traceability

1. Definition

Requirement Traceability is the ability to track and link each requirement throughout the software development life cycle (SDLC) — from its origin (stakeholder/business need) through its implementation (design, code) and validation (testing), and even to its maintenance.

It ensures that every requirement is accounted for, implemented, tested, and aligned with business objectives.

2. Importance of Requirement Traceability

- Ensures that no requirement is lost, forgotten, or misunderstood.
- Helps in impact analysis when requirements change.
- Validates that the final system satisfies the intended needs.
- Maintains consistency across different artifacts (requirements, design, code, tests).
- Supports compliance with standards and audits (important in critical domains like healthcare, aviation, defense).

3. Types of Requirement Traceability

Forward Traceability

- Tracking requirements from their source through design, development, and testing.
- Helps ensure that each requirement is implemented and tested.
- **Example:** Requirement "System shall support login with OTP" → mapped to Design (UI + backend) → mapped to Test Case (OTP verification test).

Backward (Reverse) Traceability

- Tracking from implementation/testing back to requirements.
- Ensures that every part of the system has a justification and relates to a business need (avoiding "gold plating").

Created with formatmagic.ai - One click automated formatting

- **Example:** A feature in the code should link back to a requirement. If not, it may be unnecessary.

Bi-directional Traceability

- Combines forward and backward traceability.
- Ensures full visibility in both directions.
- **Example:** A requirement is linked to test cases (forward), and every test case is linked back to a requirement (backward).

4. Requirement Traceability Matrix (RTM)

A document or table used to track the mapping between requirements and other project elements.

Structure of RTM:

REQUIREMENT ID	REQUIREMENT DESCRIPTION	DESIGN REFERENCE	CODE MODULE	TEST CASE ID	STATUS
RQ-001	User login with OTP	UI-Login, DB-Auth	auth.js	TC-101	Passed
RQ-002	Password reset feature	UI-Reset, DB-User	reset.js	TC-105	Pending

Purpose:

- Ensures coverage of all requirements.
- Makes it easy to see which requirements are implemented & tested.
- Helps in change management (quickly identifies impacted components).

5. Benefits of Requirement Traceability

- **Completeness:** All requirements are considered.
- **Quality:** Enhances verification & validation process.
- **Change Control:** Easy to analyze the effect of requirement changes.
- **Transparency:** Provides clarity to stakeholders.
- **Compliance:** Satisfies standards like ISO, IEEE, CMMI, DO-178C.

Created with formatmagic.ai - One click automated formatting

6. Challenges in Requirement Traceability

- Maintaining links in large projects is time-consuming.
- Risk of inconsistent updates if requirements change frequently.
- Requires specialized tools (like IBM DOORS, JIRA, ReqView).
- Sometimes developers/testers may ignore traceability due to tight deadlines.

7. Tools for Requirement Traceability

- JIRA with plugins (for Agile projects).
- IBM Rational DOORS (industry standard).
- HP ALM / Quality Center.
- ReqView, Polarion.

8. Example Scenario

Suppose a bank is developing an Online Banking System.

Requirement: "Customer must be able to transfer funds between accounts."

Traceability links:

- **Design:** Money Transfer UI + Transaction Service.
- **Code:** transferFunds.java.
- **Test Case:** TC-220 "Verify fund transfer between accounts".
- **Validation:** Demonstrated to the client.

If the requirement changes to include cross-bank transfers, the traceability matrix helps identify which design modules, code files, and test cases need updating.

9. Summary (Exam Point of View)

- Requirement Traceability is the practice of maintaining a clear mapping of requirements across all stages of development.
- It has forward, backward, and bi-directional types.
- The Requirement Traceability Matrix (RTM) is the main tool used.
- It ensures completeness, correctness, and alignment with stakeholder needs.
- Helps in impact analysis, testing coverage, and compliance.

Created with formatmagic.ai - One click automated formatting

Requirement Prioritization

1. Definition

Requirement prioritization is the process of ranking and organizing requirements based on their importance, value, and urgency so that the development team can decide which requirements should be implemented first, later, or possibly excluded.

In simple words: Since resources (time, budget, manpower) are limited, not all requirements can be implemented at once. Prioritization helps in identifying the "must-have" vs. "nice-to-have" requirements.

2. Why Requirement Prioritization is Needed?

- Limited resources – projects usually have time and budget constraints.
- Stakeholder conflicts – different stakeholders value different requirements.
- Incremental/iterative development – in Agile or phased models, only top-priority features are delivered early.
- Risk reduction – high-risk or business-critical requirements are handled earlier.
- Customer satisfaction – ensures that features most important to users are delivered first.

📌 **Example:** In an online shopping system:

- **High priority:** Secure payment system, product browsing, and cart.
- **Low priority:** Theme customization or product recommendation AI.

3. Factors Affecting Prioritization

- Business Value – Does the requirement provide revenue or competitive advantage?
- Cost/Effort – Is it expensive or time-consuming to implement?
- Risk – Will ignoring this requirement introduce technical or business risk?
- Stakeholder Needs – How important is it to key stakeholders?
- Dependencies – Some requirements cannot be implemented before others.
- Legal/Regulatory Compliance – Requirements necessary for law compliance get top priority.

Created with formatmagic.ai - One click automated formatting

4. Techniques for Requirement Prioritization

i. MoSCoW Method

- Must Have – critical for the system to function (e.g., login in banking system).
- Should Have – important but not vital (e.g., mobile-friendly UI).
- Could Have – desirable if resources allow (e.g., dark mode).
- Won't Have (for now) – excluded for this version (e.g., VR shopping).

ii. Kano Model

Classifies requirements into:

- Basic Needs – must be fulfilled, otherwise dissatisfaction (e.g., secure login).
- Performance Needs – more implementation = more satisfaction (e.g., faster checkout).
- Delighters – unexpected features that surprise customers (e.g., free shipping).

iii. Pairwise Comparison

Requirements are compared two at a time and ranked by importance.

Example: Between "Live chat support" and "AI recommendations", which one is more important?

iv. 100 Dollar Method

Stakeholders are given "virtual money" ($100) to distribute among requirements.

The requirements with the most "investment" are top priority.

v. Analytic Hierarchy Process (AHP)

A mathematical approach that assigns weights and scores to requirements.

Used for complex projects with many conflicting requirements.

5. Challenges in Requirement Prioritization

- Conflicting stakeholder interests – different stakeholders have different priorities.

Created with formatmagic.ai - One click automated formatting

- Changing business needs – priorities may shift during project execution.
- Large number of requirements – difficult to evaluate each requirement.
- Limited time for decision-making – quick prioritization may lead to poor choices.

6. Best Practices

- Involve all stakeholders to avoid conflicts later.
- Use a structured technique (MoSCoW, Kano, AHP) instead of subjective opinions.
- Revisit prioritization regularly because requirements evolve over time.
- Consider both short-term goals (MVP release) and long-term vision.

7. Real-World Example

In developing WhatsApp, the prioritization might look like:

- **Must Have:** Messaging, contacts sync, encryption.
- **Should Have:** Voice calls.
- **Could Have:** Stickers & wallpapers.
- **Won't Have (initially):** Payments feature.

This is why WhatsApp launched first as a messaging-only app, then later added calls, and much later introduced payments.

✅ **Summary (Exam-Ready Line):**

Requirement prioritization is the process of ranking requirements based on value, risk, cost, and stakeholder needs. Techniques like MoSCoW, Kano Model, Pairwise Comparison, 100 Dollar Method, and AHP are commonly used to ensure critical requirements are delivered first within limited resources.

Created with formatmagic.ai - One click automated formatting

Trade-off Analysis

1. Introduction

In Software Requirement Engineering, trade-off analysis is the process of evaluating different requirements, solutions, or design alternatives to balance competing factors such as cost, time, quality, performance, and customer satisfaction.

Since not all requirements can be fulfilled equally (due to limited budget, technology, or resources), engineers must make compromises to select the most feasible option.

Trade-off analysis is also called decision analysis because it helps in choosing between multiple conflicting options.

2. Need for Trade-off Analysis

- Software projects often face conflicting requirements.
- **Example:** A client wants high security and fast performance simultaneously, but strong security may slow down performance.
- Limited resources (budget, time, manpower) force organizations to prioritize some requirements over others.
- Ensures that the final system delivers maximum business value while staying feasible.

3. Dimensions of Trade-offs

When performing trade-off analysis, engineers consider multiple dimensions:

Cost vs. Quality

Higher quality requires more cost (better tools, testing, skilled developers).

Trade-off: Accept slightly lower quality to reduce costs.

Time vs. Functionality

Delivering all features may take longer.

Created with formatmagic.ai - One click automated formatting

Trade-off: Release core features early, delay non-critical ones (Agile MVP approach).

Performance vs. Security

Strong encryption improves security but reduces system performance.

Trade-off: Balance optimal performance with acceptable security.

Flexibility vs. Simplicity

Flexible systems are more complex.

Trade-off: Simpler design with limited flexibility or complex design with long-term adaptability.

4. Steps in Trade-off Analysis

Identify Conflicting Requirements

List out all requirements that clash (e.g., low cost vs. high performance).

Define Evaluation Criteria

Set measurable factors such as cost, time, risk, usability, security.

Assign Weights to Criteria

Give priority weightage based on stakeholder needs.

Example: Cost (40%), Performance (30%), Security (20%), Usability (10%).

Generate Alternatives

Create possible design or implementation solutions.

Evaluate Alternatives

Compare alternatives using decision-making techniques (e.g., weighted scoring model, cost-benefit analysis).

Select Optimal Trade-off

Choose the option that provides maximum value with acceptable compromises.

Created with formatmagic.ai - One click automated formatting

5. Techniques for Trade-off Analysis

- Weighted Scoring Model – Assign scores to alternatives against each criterion.
- Cost-Benefit Analysis – Compare estimated benefits with costs.
- SWOT Analysis – Identify Strengths, Weaknesses, Opportunities, Threats.
- Analytic Hierarchy Process (AHP) – Structured method to rank alternatives.
- Pareto Analysis (80/20 Rule) – Focus on 20% of requirements that deliver 80% of the value.

6. Example

Suppose a company is developing an online banking system with these requirements:

- High security (data encryption, multi-factor authentication).
- Fast performance (quick transactions).
- Low cost (budget constraints).

Conflict:

- High security increases cost and reduces performance.
- Low cost reduces quality and features.

Trade-off Decision:

- Implement moderate security (2FA, but not very expensive biometrics).
- Optimize performance by using efficient algorithms.
- Accept slightly higher cost to ensure customer trust.

Thus, the optimal balance is achieved.

7. Importance of Trade-off Analysis

- Ensures rational decision-making.
- Aligns system development with business goals.
- Reduces conflicts between stakeholders.
- Prevents project delays caused by unrealistic requirements.
- Helps in risk management by identifying acceptable compromises.

Created with formatmagic.ai - One click automated formatting

8. Short Summary (Exam-Ready)

Trade-off analysis in SRE is the process of balancing conflicting requirements such as cost, quality, time, and performance. Since not all requirements can be met fully, compromises must be made to achieve the best possible solution within constraints. It involves identifying conflicts, setting evaluation criteria, generating alternatives, and selecting the most feasible option using techniques like cost-benefit analysis, weighted scoring, and SWOT analysis.

👉 **Example:** In an online banking system, a trade-off may be made between high security, low cost, and fast performance to deliver an optimal solution.

Created with formatmagic.ai - One click automated formatting

Risk Analysis and Impact Analysis

1. Introduction

In Software Requirement Engineering, risks are potential problems that might occur in the future and negatively affect the software project. Risk analysis focuses on identifying, assessing, and managing risks, while impact analysis is the process of evaluating the consequences of changes in requirements on the system and project.

Both activities are crucial to ensure the project remains feasible, stable, and reliable throughout its lifecycle.

2. Risk Analysis

2.1 Definition

Risk Analysis is the systematic process of identifying risks, analyzing their likelihood and consequences, and planning how to manage them in a software project.

2.2 Steps in Risk Analysis

Risk Identification

Recognizing potential problems that could affect requirements or project success.

Examples: unclear requirements, technology failure, unrealistic deadlines.

Risk Classification

- Project Risks → affect schedule, budget, resources.
- Product Risks → affect quality, functionality, performance.
- Business Risks → affect overall success (market acceptance, ROI).

Risk Assessment (Estimation)

Determining:

- **Probability (likelihood of occurrence:** low/medium/high).
- **Impact/Severity (consequence:** minor/major/critical).

Created with formatmagic.ai - One click automated formatting

Risk Prioritization

Ranking risks based on risk exposure (Probability × Impact).

Focus on high-priority risks first.

Risk Mitigation and Planning

Developing strategies to avoid, reduce, transfer, or accept risks.

2.3 Example of Risk Analysis

RISK	PROBABILITY	IMPACT	EXPOSURE	MITIGATION STRATEGY
AMBIGUOUS REQUIREMENTS	High	High	Critical	Conduct detailed requirement workshops
NEW TECHNOLOGY ADOPTION	Medium	High	Major	Provide training, hire experts
TIGHT DEADLINE	High	Major	Critical	Adjust scope, add more resources
MARKET CHANGE	Low	Critical	Major	Regular stakeholder feedback

3. Impact Analysis

3.1 Definition

Impact Analysis is the process of evaluating the consequences of a proposed change in requirements on the system's architecture, design, cost, schedule, and other requirements.

3.2 Purpose of Impact Analysis

- To estimate effort, cost, and risk of implementing a change.
- To understand the ripple effect of changes across requirements.
- To prevent uncontrolled changes (requirements creep).
- To maintain consistency and stability in the project.

Created with formatmagic.ai - One click automated formatting

3.3 Steps in Impact Analysis

Change Request Identification

A stakeholder suggests a change in requirement.

Example: “Add biometric login instead of just password login.”

Traceability Check

Use requirement traceability matrices to identify which parts of the system will be affected (design, modules, tests, documents).

Dependency Analysis

Analyze interdependencies between requirements (functional, non-functional, business).

Impact Assessment

Evaluate impact on:

- Cost (extra resources, licenses).
- Schedule (delays due to redesign).
- Quality (security, performance).
- Stakeholders (usability, satisfaction).

Decision Making

Accept, modify, or reject the change based on analysis results.

3.4 Example of Impact Analysis

Change Requested: Add multi-language support in an e-commerce website.

Affected Modules: UI design, database (product descriptions), backend APIs, testing.

Impact on Schedule: +3 weeks development time.

Impact on Cost: $15,000 increase (translation + testing).

Impact on Quality: Improved customer satisfaction (wider audience).

Decision: Approved due to high business value.

Created with formatmagic.ai - One click automated formatting

4. Relationship between Risk Analysis and Impact Analysis

Risk Analysis looks forward → What could go wrong in the future?

Impact Analysis looks after a change request → What will be affected if we apply this change?

Together, they improve project decision-making, reduce uncertainty, and ensure requirements remain realistic and manageable.

5. Key Points for Exams

- Risk = uncertainty that can harm the project.
- Risk Analysis = identifying, assessing, and managing risks.
- Impact Analysis = studying the consequences of requirement changes.
- Both ensure stability, quality, and feasibility of software projects.

✅ **Summary (Exam-Ready):**

Risk analysis identifies potential problems that may threaten a project, classifies them, and develops strategies to manage them. It involves risk identification, classification, assessment, and mitigation. Impact analysis, on the other hand, evaluates the consequences of changes in requirements, estimating their effect on cost, schedule, and quality. While risk analysis prevents failures by managing uncertainties, impact analysis ensures requirement changes are handled systematically. Together, they help in delivering high-quality software on time and within budget.

Created with formatmagic.ai - One click automated formatting

Requirement Management

Definition

Requirement Management is the systematic process of documenting, organizing, analyzing, tracing, prioritizing, and controlling requirements throughout the software development lifecycle. Its main goal is to ensure that requirements are clear, feasible, aligned with business objectives, and adaptable to changes.

1. Purpose of Requirement Management

- To maintain a clear understanding of requirements among stakeholders.
- To handle changes in requirements during development.
- To ensure requirements traceability across all stages.
- To reduce risks by identifying conflicts, ambiguities, or missing requirements early.
- To make sure the final system meets customer needs.

📌 **Example:** In a mobile banking app project, requirement management ensures that a late change like “adding fingerprint login” does not conflict with earlier requirements and is properly documented, approved, and tested.

2. Activities in Requirement Management

Requirement management involves several core activities:

a. Requirement Documentation

- All requirements are written in a structured document (SRS – Software Requirement Specification).
- Ensures requirements are clear, unambiguous, and testable.

b. Requirement Identification

- Each requirement is assigned a unique identifier (like RQ-101, RQ-102).
- Helps in tracking and referencing requirements throughout the project.

Created with formatmagic.ai - One click automated formatting

c. Requirement Traceability

- Linking each requirement to design, code, and test cases.
- Helps in verifying that no requirement is left unimplemented.

d. Requirement Prioritization

- Ranking requirements as High, Medium, Low priority based on business value and technical feasibility.

e. Requirement Change Management

- Handling changes requested by stakeholders.
- Ensures change is analyzed for impact, cost, risk, and schedule before approval.

f. Requirement Version Control

- Maintaining different versions of requirements as they evolve.
- Prevents confusion when multiple stakeholders are working with different documents.

3. Requirement Management Process

- Gather requirements (elicitation).
- Analyze and document them.
- Assign identifiers and organize.
- Prioritize requirements.
- Establish traceability links.
- Manage changes (with approval workflows).
- Review and validate requirements with stakeholders.
- Maintain updated documentation until project completion.

4. Tools for Requirement Management

Several tools support automation of requirement management:

- IBM DOORS
- Jama Connect
- HP ALM (Application Lifecycle Management)
- Jira + Confluence

Created with formatmagic.ai - One click automated formatting

- ReqView

📌 **Example:** Using Jira, each requirement is logged as a ticket, linked with tasks, and marked with status (To-do, In progress, Done). This makes tracking easy.

5. Importance of Requirement Management

- Avoids scope creep (uncontrolled expansion of requirements).
- Ensures alignment with business goals.
- Reduces misunderstanding between stakeholders and developers.
- Improves quality assurance through traceability and validation.
- Provides control over changes, saving time and cost.

📌 **Example:** In e-commerce software, poor requirement management may lead to missing features like "cart saving for later," frustrating users and causing project failure.

6. Challenges in Requirement Management

- Frequent changes in requirements.
- Conflicting stakeholder needs.
- Lack of proper documentation.
- Poor communication between business and technical teams.
- Managing requirements in large, complex projects.

7. Best Practices

- Use structured requirement documents (like IEEE SRS standard).
- Involve all stakeholders during reviews.
- Apply version control and maintain history.
- Use requirement management tools for automation.
- Conduct impact analysis before approving changes.

Created with formatmagic.ai - One click automated formatting

Summary

Requirement Management is the discipline of controlling and organizing requirements throughout the software project. It covers documentation, identification, traceability, prioritization, change management, and validation. Without proper management, requirements may become inconsistent, incomplete, or unachievable, leading to project delays and failures.

📌 **In exams, you should explain:**

- Definition
- Purpose
- Activities (documentation, traceability, change management, etc.)
- Process steps
- Importance + Challenges
- Example

Created with formatmagic.ai - One click automated formatting

Interaction Between Requirement and Architecture

1. Introduction

Requirements and software architecture are closely linked.

Requirements describe what the system should do, while architecture defines how the system will be structured and designed to meet those requirements.

The interaction is bidirectional:

- Requirements influence the architecture.
- Architecture can also shape or refine requirements.

2. Relationship Between Requirements and Architecture

Requirements Influence Architecture

Functional requirements decide the main components and their interactions.

Non-functional requirements (like performance, scalability, and security) strongly guide architectural choices.

Example: If a system requires high availability, the architecture might include load balancers, redundancy, and failover mechanisms.

Architecture Influences Requirements

Architectural constraints can refine or modify requirements.

Example: If the architecture is cloud-based, requirements may evolve to include multi-region deployment or data synchronization.

Certain technologies or frameworks may limit or enhance features, requiring adjustments in requirements.

Created with formatmagic.ai - One click automated formatting

3. Key Areas of Interaction

Functional Requirements ↔ Architecture

These define the main modules, subsystems, and workflows.

Example: An e-commerce system requires a shopping cart, payment gateway, and order tracking, which shape the core architecture.

Non-Functional Requirements ↔ Architecture

- Performance → influences database design and caching strategies.
- Security → leads to authentication layers, firewalls, and encryption mechanisms.
- Scalability → requires microservices, cloud infrastructure, or distributed systems.

Constraints

Requirements may include budget, time, or technology constraints.

These constraints heavily restrict architectural choices.

Example: If a project must be built in 3 months, a lightweight framework (like Django or Spring Boot) may be chosen instead of developing a custom solution.

4. Importance of Aligning Requirements with Architecture

- Ensures that the system design meets customer expectations.
- Prevents rework and costly redesigns later in development.
- Helps balance trade-offs between conflicting requirements (e.g., performance vs. security).
- Improves system quality by ensuring that architecture supports required functionality and qualities.

5. Challenges in Requirement–Architecture Interaction

- Incomplete or ambiguous requirements → can misguide architectural design.
- Changing requirements (requirement evolution) → may require costly redesigns.

Created with formatmagic.ai - One click automated formatting

- Conflicting non-functional requirements (e.g., performance vs. security) → make architecture decisions difficult.
- Technology constraints → may force compromise in meeting some requirements.

6. Example Case Study

Banking System

Requirements:

- Functional → Money transfer, account management, statement generation.
- Non-functional → High security, 24/7 availability, fast transaction speed.

Architecture:

- Functional mapping → Modules for transfer, accounts, statements.
- Non-functional mapping → Secure encryption protocols, load balancers, replicated databases.

Here, requirements directly shape the architecture, while architectural decisions (e.g., using blockchain or centralized DB) may alter requirements like transaction cost or performance expectations.

7. Summary (Exam-Ready)

- Requirements and architecture are interdependent.
- Requirements → drive the architecture, while architecture → refines requirements.
- Both functional and non-functional requirements strongly influence architectural choices.
- Close alignment avoids mismatches, reduces risks, and ensures system quality.
- Challenges include ambiguity, changes, and conflicts in requirements.
- **Example:** An e-commerce or banking system clearly shows this two-way interaction

Created with formatmagic.ai - One click automated formatting

Requirement Elicitation

1. Definition

Requirement Elicitation is the process of collecting and discovering requirements from stakeholders, customers, users, and other sources to understand what the system should do.

It is the first step in Requirement Engineering and is sometimes called requirement gathering, discovery, or fact-finding.

> **Key Point:** It is not just asking stakeholders, but also involves investigation, observation, and analysis to extract hidden needs.

2. Objectives of Requirement Elicitation

- Understand the customer's needs and expectations.
- Identify functional and non-functional requirements.
- Resolve conflicts between stakeholders' needs.
- Capture constraints (budget, technology, timeline).
- Provide a clear foundation for requirement analysis and specification.

3. Inputs for Requirement Elicitation

- Stakeholders (users, customers, managers, developers).
- Business documents (manuals, reports, policies).
- Existing system (if upgrading an old system).
- Market study / competitor products.

4. Techniques of Requirement Elicitation

There are multiple techniques; choice depends on project type, size, and stakeholders.

Created with formatmagic.ai - One click automated formatting

A. Traditional Techniques

- **Interviews**
  - One-on-one discussions with stakeholders.
  - **Types:** structured, semi-structured, unstructured.
  - **Example:** Asking a bank manager what features should exist in online banking.
- **Questionnaires / Surveys**
  - Useful for collecting data from a large group.
  - **Example:** Sending forms to employees for new payroll system requirements.
- **Brainstorming**
  - A group activity to generate ideas quickly.
  - **Example:** A team brainstorming features for a new shopping app.
- **Workshops / Focus Groups**
  - Gather multiple stakeholders to discuss needs and reach consensus.
- **Observation**
  - Watching users perform tasks in real environments.
  - **Example:** Observing how a cashier uses the billing system in a supermarket.
- **Document Analysis**
  - Studying business reports, policies, and manuals to extract hidden requirements.

B. Modern / Advanced Techniques

- **Prototyping**
  - Building a simple version of the system to clarify requirements.
  - **Example:** A clickable mockup of a mobile banking app.
- **Use Case / Scenario Analysis**
  - Defining how users will interact with the system.
- **Joint Application Development (JAD)**
  - Intense workshops with users and developers to agree on requirements.
- **Ethnographic Studies**
  - Long-term observation of users in their environment.

Created with formatmagic.ai - One click automated formatting

5. Challenges in Requirement Elicitation

- Ambiguity – Users may not express requirements clearly.
- Conflicting Requirements – Different stakeholders may want different features.
- Hidden Requirements – Some needs are not explicitly stated.
- Changing Requirements – Needs may evolve over time.
- Communication Gap – Misunderstandings between users and developers.

6. Example Case Study

Hospital Management System

Techniques used:

- Interviews with doctors (to know about patient records).
- Observation of nurses (how they track patient medication).
- Document analysis (medical policies, reporting standards).
- Prototyping (demo of appointment booking system).

Result: A complete set of requirements for modules like patient records, billing, pharmacy, and doctor schedules.

7. Summary (Exam-Ready)

- Requirement Elicitation = process of discovering user needs.
- **Objectives:** identify requirements, constraints, and expectations.
- **Techniques:** interviews, surveys, brainstorming, workshops, observation, document analysis, prototyping, JAD, ethnography.
- **Challenges:** ambiguity, conflicts, hidden/changed requirements, communication gaps.
- **Example:** Hospital system / banking app where multiple techniques are applied.

Created with formatmagic.ai - One click automated formatting

Elicitation Sources and Techniques

1. Introduction

Requirement elicitation is the first and most important step in the Software Requirement Engineering (SRE) process. It means gathering, discovering, and uncovering the needs of stakeholders so that they can be transformed into well-defined requirements.

Since stakeholders may have different backgrounds, goals, and expectations, requirements cannot be "assumed." Instead, they must be carefully elicited from reliable sources using systematic techniques.

2. Elicitation Sources

Sources are the places or entities from which requirements can be obtained. They ensure that the requirements represent real-world needs.

(a) Stakeholders

Definition: People or groups who have a direct or indirect interest in the system.

Examples:

- Customers (who request the product).
- End-users (who actually use it).
- Developers (who build it).
- Managers (who fund or supervise).

Importance: Stakeholders provide both functional (what system should do) and non-functional (quality expectations like security, usability) requirements.

(b) Existing Systems or Products

Many projects are built to improve or replace an existing system.

The old system acts as a reference source.

Example:

When upgrading an ATM software, the current ATM functions provide baseline requirements.

Created with formatmagic.ai - One click automated formatting

(c) Business Documents

Include policies, contracts, company rules, and standards.

They reveal constraints and compliance requirements.

Example: A banking regulation document may require “Two-factor authentication for all online transactions.”

(d) Domain Experts

Professionals with specialized knowledge of the business or technical field.

Example: In hospital management software, doctors and nurses can highlight workflow and safety requirements.

(e) Market and Competitors

Market studies, customer feedback, and competitor products reveal user expectations.

Example: If a rival app offers “dark mode,” users may expect it in your app too.

(f) Observation of the Environment

Understanding the real-world environment where the system will operate.

Example: Observing factory workers helps in designing a machine interface suitable for noisy surroundings.

3. Elicitation Techniques

These are the methods used to collect requirements from the above sources. Choice of technique depends on project type, stakeholder availability, and complexity.

(a) Interviews

Description: One-to-one or group discussions with stakeholders.

Advantages: Provides deep insights, clarifies doubts.

Example: Asking an HR manager what features they need in a payroll system.

(b) Questionnaires/Surveys

Description: Written set of structured/unstructured questions.

Advantages: Collects data from many people quickly.

Example: Sending Google Forms to students to know what features they expect in an online learning platform.

(c) Workshops / Focus Groups

Created with formatmagic.ai - One click automated formatting

Description: Bringing together stakeholders for brainstorming.

Advantages: Promotes consensus, reduces conflicts early.

Example: A workshop with teachers, parents, and students to discuss requirements for a school management system.

(d) Observation (Job Shadowing)

Description: Analyst observes how users work in their natural environment.

Advantages: Captures real, unspoken needs.

Example: Watching how a cashier uses POS software to detect pain points.

(e) Document Analysis

Description: Studying manuals, existing reports, business rules.

Example: Reading "Hospital Safety Guidelines" before building patient record software.

(f) Prototyping

Description: Building a small working model (mock-up) of the system.

Advantages: Helps stakeholders visualize the system and refine unclear requirements.

Example: Creating a prototype of a mobile app to confirm features like navigation, search, and checkout.

(g) Brainstorming

Description: Generating a wide range of ideas quickly.

Example: Brainstorming session with marketing and IT teams to suggest possible features for a new e-commerce website.

(h) Joint Application Development (JAD)

Description: Intensive, structured workshops involving developers, clients, and users.

Advantages: Reduces requirement gaps, encourages collaboration.

(i) Ethnographic Studies

Description: Long-term immersion into the user's environment to deeply understand practices.

Example: An analyst working within a call center for a week to study challenges before designing call management software.

4. Challenges in Requirement Elicitation

- Conflicting requirements from different stakeholders.
- Unclear or vague needs ("system must be user-friendly").
- Communication barriers (technical vs. non-technical language).
- Hidden requirements (stakeholders assume some features are "obvious").

Created with formatmagic.ai - One click automated formatting

5. Example (Case Study)

Project: Building a University Student Portal.

Sources Used:

- Students → Course registration needs.
- Teachers → Attendance and grading tools.
- Administration → Fee management and compliance.
- Existing systems → Current portal problems.

Techniques Applied:

- Interviews with faculty.
- Survey of students.
- Prototyping of dashboard.
- Observation of registration counters.

Result:

A portal design that balances academic, financial, and administrative needs.

6. Summary (Exam-Ready)

- Requirement Elicitation = Discovering stakeholder needs.
- **Sources:** Stakeholders, documents, domain experts, environment, market, existing systems.
- **Techniques:** Interviews, surveys, workshops, observation, document analysis, prototyping, brainstorming, JAD, ethnography.
- Good elicitation avoids misunderstandings, saves cost, and ensures system success.

Created with formatmagic.ai - One click automated formatting

Requirement Specification and Documentation

1. Introduction

Requirement Specification and Documentation is the process of formally recording and describing software requirements in a structured, precise, and understandable way. It serves as a contract between stakeholders (clients, users) and the development team.

Without proper specification and documentation, requirements remain vague, leading to misunderstandings, scope creep, and project failure.

2. Requirement Specification

Definition

Requirement Specification is the detailed representation of software requirements that defines what the system should do and sometimes how it should behave under certain conditions.

It transforms user needs into clear, measurable, and testable requirements.

Purpose

- To remove ambiguity and misinterpretations.
- To act as a reference throughout the software lifecycle.
- To help in validation, verification, and testing.
- To serve as the baseline for project planning and cost estimation.

3. Types of Requirement Specification

a) Functional Requirement Specification (FRS)

- Describes what the system should do.
- Includes inputs, outputs, processes, and interactions.

Example:

“The system shall allow users to log in using email and password.”

Created with formatmagic.ai - One click automated formatting

“The ATM shall dispense cash only if the balance is sufficient.”

b) Non-Functional Requirement Specification (NFRS)

- Defines quality attributes, performance, usability, security, etc.
- Describes how well the system performs its functions.

Example:

“The system should support 1000 concurrent users.”

“The response time must be less than 2 seconds.”

c) Interface Requirement Specification

Defines how the system will interact with other systems, hardware, or users.

Example:

“The application must integrate with a payment gateway via API.”

d) Domain Requirement Specification

Specific to the domain (finance, healthcare, education, etc.).

Example:

“The hospital management system must comply with HIPAA regulations.”

4. Requirement Specification Techniques

Natural Language (NL) Specification

- Written in plain text.
- Easy for stakeholders to understand.
- **Risk:** Ambiguity due to lack of precision.

Structured Language Specification

- Uses standard templates, keywords, and controlled vocabulary.
- **Example:** IEEE SRS format.

Mathematical/ Formal Specification

- Uses formal languages (Z notation, VDM).

Created with formatmagic.ai - One click automated formatting

- Precise and unambiguous but difficult for non-technical stakeholders.

Graphical Specification

- Uses diagrams (UML, flowcharts, data flow diagrams).
- Easy to understand visually.

5. Documentation of Requirements

Definition

Requirement Documentation is the systematic process of recording, organizing, and maintaining all requirements in a structured document, usually called SRS (Software Requirement Specification).

Importance of Documentation

- Provides a single reference point for all stakeholders.
- Acts as a baseline for design and development.
- Helps in traceability and change management.
- Ensures compliance with standards (e.g., IEEE 830/29148).

6. Components of SRS (IEEE Standard 830/29148)

- Introduction
- Purpose, scope, definitions, references.
- Overall Description
- Product perspective, user needs, constraints, assumptions.
- Functional Requirements
- Use cases, system features, inputs/outputs.
- Non-Functional Requirements
- Performance, usability, reliability, scalability.
- System Interfaces
- External APIs, hardware, software.
- Constraints
- Business rules, legal, regulatory.
- Appendices
- Glossary, references, supporting material.

Created with formatmagic.ai - One click automated formatting

7. Best Practices for Specification & Documentation

- Use clear, simple, and unambiguous language.
- Ensure completeness (no missing requirements).
- Maintain consistency (no conflicting requirements).
- Prioritize requirements based on importance.
- Use diagrams and models for better visualization.
- Involve stakeholders in reviews to avoid misinterpretation.

8. Example of Requirement Specification (ATM System)

Functional Requirement:

“The ATM shall allow the user to withdraw cash in multiples of 500.”

Non-Functional Requirement:

“The ATM shall process each transaction within 10 seconds.”

Interface Requirement:

“The ATM shall communicate with the bank server via secure HTTPS.”

9. Conclusion

Requirement Specification and Documentation is a critical step in Software Requirement Engineering. It ensures that all stakeholders share a common understanding of what the system will deliver. A well-prepared SRS document acts as a blueprint for development, testing, and maintenance, ultimately reducing risks and ensuring project success.

Created with formatmagic.ai - One click automated formatting

Specification Sources and Techniques

1. Introduction

Requirement Specification is the process of documenting the requirements in a clear, structured, and understandable format. It defines what the system should do and provides the foundation for design, development, and validation.

To prepare high-quality specifications, engineers rely on sources (where requirements come from) and techniques (methods to gather, write, and structure them).

2. Specification Sources

Specification sources are the origins of requirement information. They provide input for preparing the Software Requirements Specification (SRS).

(a) Customer / Client

- Main source of requirements.
- Provides business goals, objectives, and needs.

Example: A client may say, “The system should allow online payment through credit card and PayPal.”

(b) End Users

- Users provide practical insights into how they expect to interact with the system.

Example: Cashiers using POS software may request quick shortcuts for billing.

(c) Business Rules & Policies

- Existing organizational rules, policies, and industry regulations act as sources.

Example: A banking system must follow KYC (Know Your Customer) regulations.

(d) Existing Systems / Legacy Systems

- Old systems often inspire requirements for new ones.

Created with formatmagic.ai - One click automated formatting

Example: A hospital upgrading from a manual record system may require digital patient history.

(e) Domain Experts

- Provide specialized knowledge about the industry.

Example: In aviation software, a pilot or air traffic controller may define operational needs.

(f) Market and Competitors

- Requirements may come from market demand or competitor features.

Example: If competitor e-commerce sites offer one-click checkout, the requirement may be included.

(g) Standards and Regulations

- Government and industry standards guide requirements.

Example: GDPR compliance for software handling European customer data.

3. Specification Techniques

Different techniques help transform sources into formal, well-documented requirements.

(a) Natural Language Specification

- Writing requirements in simple, structured English.

Example: “The system shall allow users to reset their password via email.”

- **Pros:** Easy to understand.
- **Cons:** Ambiguous if not structured properly.

(b) Structured Language / Templates

- Use of requirement templates or predefined sentence structures to avoid ambiguity.

Example:

“The system shall .”

Created with formatmagic.ai - One click automated formatting

“The system shall display order details within 3 seconds after the user clicks the button.”

(c) Use Cases

- Describe interaction between actors (users or systems) and the system.

Example: Use Case: “Place Order” → Actor: Customer → System: Validates cart → Confirms payment → Generates invoice.

(d) User Stories

- **Short, simple statements in the format:**

“As a , I want , so that .”

Example: “As a student, I want to upload assignments online, so that I don’t have to submit them physically.”

(e) Models and Diagrams

- UML diagrams (class diagrams, sequence diagrams, state charts, ER diagrams).

Example: Activity Diagram showing steps of online payment.

(f) Prototyping

- Building mockups or prototypes to clarify requirements.

Example: Creating a UI prototype for a mobile app before actual development.

- **Benefit:** Helps clients visualize the product.

(g) Formal Specification Techniques

- Use of mathematical/logical notations for critical systems.

Example: In aviation software, formal methods ensure correctness.

- **Benefit:** Eliminates ambiguity.
- **Drawback:** Requires specialized expertise.

(h) Scenarios

- Real-life stories describing how the system will be used.

Created with formatmagic.ai - One click automated formatting

Example: “A customer places an order, the system verifies stock, sends confirmation, and arranges delivery.”

4. Example (Banking System)

Imagine creating a specification for Internet Banking System:

Sources: Client → wants money transfer, End Users → want mobile-friendly interface, Regulations → need KYC & two-factor authentication.

Techniques: Use Cases (money transfer process), Prototyping (mobile app UI), Structured Templates (security requirements).

5. Conclusion

- Sources provide raw material for requirements.
- Techniques ensure they are properly written, clear, and structured.
- Good specifications = fewer misunderstandings, reduced rework, and better software quality.

Created with formatmagic.ai - One click automated formatting

Requirements Validation and Techniques

1. Introduction to Requirements Validation

Requirements validation is the process of ensuring that the documented requirements accurately reflect the needs of the stakeholders, are complete, consistent, and feasible, and that they provide a solid foundation for design and development.

It answers the question:

👉 “Are we building the right system?” (validation)

while verification answers:

👉 “Are we building the system right?”

Validation ensures that:

- Requirements are aligned with user goals.
- Requirements are realistic and implementable.
- Errors in requirements are detected early, saving cost and time.

📌 **Example:** If stakeholders need an online payment system, but the requirement document only states “The system shall allow transactions”, validation will identify that this requirement is vague, incomplete, and not testable.

2. Objectives of Requirements Validation

- Ensure requirements reflect real customer needs.
- Detect errors, ambiguities, or omissions before design begins.
- Confirm that requirements are testable and measurable.
- Ensure consistency across all requirements.
- Gain stakeholder agreement and approval.

3. Common Requirement Errors Detected in Validation

- **Ambiguity** – Requirement has multiple interpretations.
  - **Example:** “System should be fast” (what is fast? 2 seconds or 10 seconds?).

Created with formatmagic.ai - One click automated formatting

- **Incompleteness** – Missing functionality or constraints.
    - **Example:** Requirement for “user login” but no mention of password reset.
- **Inconsistency** – Conflicting requirements.
    - **Example:** One requirement says “password length = 6 characters” and another says “password length ≥ 8 characters”.
- **Unrealistic requirements** – Beyond technical or budget limits.
    - **Example:** “System must support 1 billion users at launch”.
- **Unverifiable requirements** – Cannot be tested objectively.
    - **Example:** “System should be user-friendly”.

4. Techniques of Requirements Validation

1. Reviews

Most common validation technique.

Involves manual examination of requirements by stakeholders, developers, and testers.

Types of Reviews:

- Informal Review – quick checks by peers.
- Formal Review / Inspections – structured meetings with checklist.
- Walkthroughs – author explains requirements to peers.

📌 **Example:** A review team may use a checklist to see if all requirements are testable, consistent, and complete.

2. Prototyping

Developing a working model or mockup of the system to clarify vague requirements.

Helps stakeholders visualize the system before full development.

📌 **Example:** For an e-commerce site, a clickable prototype of the checkout process can reveal missing requirements (like applying discount codes).

3. Model Validation

Using diagrams, simulations, or models (e.g., UML, data flow diagrams) to ensure requirements match system behavior.

Created with formatmagic.ai - One click automated formatting

Useful for complex systems.

📌 **Example:** Sequence diagrams can reveal if a required interaction (e.g., order confirmation email) is missing.

4. Test-case Generation

Creating test cases from requirements to check their testability.

If a requirement cannot produce a clear test case, it is likely ambiguous or incomplete.

📌 **Example:** Requirement: "System shall display the dashboard in 3 seconds after login."

Test case: Measure page load time after login → must be ≤ 3 seconds.

5. Stakeholder Workshops / Joint Application Design (JAD)

Sessions involving customers, users, and developers to discuss, review, and agree on requirements.

Encourages collaboration and resolves conflicts.

6. Automated Consistency and Completeness Checking

Using tools to detect missing requirements, conflicts, or redundancy in requirement documents.

5. Steps in Requirements Validation Process

- Preparation – Collect requirements document, checklists, and models.
- Review & Analysis – Identify ambiguities, conflicts, omissions.
- Prototyping/Modeling (if needed) – Clarify unclear requirements.
- Stakeholder Confirmation – Present findings, confirm agreement.
- Correction & Update – Revise requirements document.

6. Benefits of Requirements Validation

- Reduces development cost by catching errors early.
- Prevents project failure due to misunderstood requirements.
- Ensures high customer satisfaction.
- Improves communication between developers and stakeholders.

Created with formatmagic.ai - One click automated formatting

> 📌 **Real-world Example:** NASA lost the Mars Climate Orbiter (1999) because requirements for measurement units (metric vs. imperial) were not validated properly. A small inconsistency caused a $327 million failure.

7. Summary (Exam-Ready)

- Requirements validation ensures we are building the right system.
- **Main goals:** correctness, completeness, consistency, realism, and testability.
- Techniques include reviews, prototyping, model validation, test-case generation, workshops, and automated tools.
- Catching requirement errors early saves cost, time, and prevents failure.
- Real-life failures (like NASA's orbiter) show why requirement validation is critical.

Created with formatmagic.ai - One click automated formatting

Management of Requirements

1. Definition

Requirement Management is the systematic process of documenting, analyzing, tracing, prioritizing, and controlling requirements throughout the software development lifecycle.

It ensures that all requirements are properly captured, updated, and aligned with the project's objectives, even as they evolve over time.

2. Purpose of Requirement Management

- To maintain consistency between requirements and the final system.
- To handle changes effectively without causing project failures.
- To ensure that stakeholders' needs are met.
- To provide a clear audit trail of how requirements are defined, modified, and implemented.
- To avoid scope creep (uncontrolled addition of requirements).

3. Key Activities in Requirement Management

(a) Requirement Identification

Assigning each requirement a unique ID for easy reference.

Example: REQ-01: System shall support 2FA login.

(b) Requirement Baseline

A baseline is the approved version of requirements at a specific point in time.

It acts as a reference point to compare future changes.

(c) Requirement Traceability

Establishing links between requirements and related artifacts (design, test cases, implementation).

Example: REQ-01 → linked with TestCase-01 and Module-Login.

Created with formatmagic.ai - One click automated formatting

(d) Requirement Change Management

Defining a process to request, evaluate, approve, or reject changes.

Example: If a client asks to add fingerprint login, a Change Request (CR) is created and analyzed before implementation.

(e) Requirement Version Control

Maintaining versions of requirements to track their history.

Example: REQ-01 v1.0 (Password + OTP), updated to REQ-01 v2.0 (Password + OTP + Fingerprint).

(f) Requirement Prioritization

Deciding which requirements are more critical and must be implemented first.

Techniques: MoSCoW (Must have, Should have, Could have, Won't have).

(g) Requirement Monitoring

Regularly tracking the status of requirements:

- Implemented
- Pending
- Under Review
- Tested

4. Tools for Requirement Management

- Jama Software
- IBM Rational DOORS
- Jira
- Azure DevOps
- Helix RM

These tools help maintain traceability, change management, and documentation.

5. Challenges in Requirement Management

- Changing requirements from stakeholders.
- Conflicts among different stakeholder needs.

Created with formatmagic.ai - One click automated formatting

- Inadequate documentation.
- Poor communication between teams.
- Difficulty in tracking dependencies across modules.

6. Best Practices

- Always document requirements clearly and get stakeholder approval.
- Use traceability matrices to connect requirements with design & testing.
- Apply change control boards (CCB) for handling requirement changes.
- Keep requirements in a central repository for easy access.
- Regularly review and validate requirements with stakeholders.

7. Example – Requirement Management in a Banking App

- **Requirement:** “System shall support biometric login.”
- **ID:** REQ-05
- **Baseline Version:** v1.0 (supports only OTP)
- **Change Request:** Client requests biometric login.
- **Updated Version:** v2.0 (supports OTP + Fingerprint + Face ID).
- **Traceability:** REQ-05 linked to Login Module + TestCase 102.
- **Monitoring:** Status = Implemented and Tested.

8. Summary

Requirement Management is about keeping control of requirements from start to end. It covers identification, documentation, traceability, change control, and monitoring. Proper management ensures that requirements are implemented correctly, adapted to changes, and deliver value to stakeholders while reducing risks of failure.

Created with formatmagic.ai - One click automated formatting

Introduction to Management

1. Definition

Management is the process of planning, organizing, leading, and controlling resources (human, financial, physical, informational) to achieve specific goals effectively and efficiently.

Effectiveness = achieving goals (doing the right things).

Efficiency = using minimum resources (doing things right).

👉 **Example:** In a software company, management ensures the project is delivered on time (effectiveness) while staying within budget (efficiency).

2. Nature of Management

- Universal – applied in all organizations (business, government, education).
- Goal-Oriented – focused on achieving objectives.
- Continuous Process – ongoing, not one-time.
- Multidisciplinary – uses knowledge from economics, psychology, sociology, etc.
- Dynamic – adapts to environmental and technological changes.

3. Levels of Management

Top-Level Management (Strategic Level)

- **Roles:** CEO, Directors, Board Members.
- **Functions:** Long-term planning, policies, overall direction.
- **Example:** Deciding to expand company services to cloud computing.

Middle-Level Management (Tactical Level)

- **Roles:** Department Heads, Project Managers.
- **Functions:** Translate top-level plans into departmental goals, supervise lower managers.
- **Example:** Project manager allocates tasks to development teams.

Created with formatmagic.ai - One click automated formatting

Lower-Level Management (Operational Level)

- **Roles:** Supervisors, Team Leaders.
- **Functions:** Day-to-day operations, directly manage workers.
- **Example:** A team lead assigns coding tasks to developers.

4. Functions of Management (Fayol's Principles)

Planning

Deciding what to do, when, how, and who will do it.

- **Example:** Planning a software release roadmap.

Organizing

Arranging resources and tasks in a structured way.

- **Example:** Dividing project team into frontend, backend, and QA.

Leading (Directing)

Guiding and motivating employees to work towards goals.

- **Example:** A manager inspires developers by rewarding performance.

Controlling

Monitoring performance, comparing with standards, and making corrections.

- **Example:** Tracking project deadlines and adjusting timelines if delayed.

5. Importance of Management

- Achieves organizational goals in time.
- Increases efficiency by minimizing waste of resources.
- Maintains discipline and teamwork.
- Handles change and innovation in a competitive environment.
- Improves decision-making with structured planning.
- Ensures sustainability and long-term growth.

Created with formatmagic.ai - One click automated formatting

6. Skills Required for Management

- Technical Skills – job-related knowledge (e.g., coding, engineering).
- Human Skills – communication, leadership, teamwork.
- Conceptual Skills – strategic thinking, problem-solving.

7. Example – Management in IT Company

- **Top-Level:** CEO decides to launch a new AI product.
- **Middle-Level:** Project managers divide teams into AI Model Training, Backend, and UI.
- **Lower-Level:** Team leads assign daily coding, testing, and debugging tasks.
- **Management Functions:** Plans roadmap → Organizes teams → Leads employees → Controls progress.

8. Summary

Management is the art of getting things done through people and resources. It ensures efficiency and effectiveness by following four main functions: Planning, Organizing, Leading, and Controlling. Without management, organizations would face confusion, inefficiency, and failure to achieve goals.

Created with formatmagic.ai - One click automated formatting

Requirements Management Problems

Requirements management is the process of documenting, analyzing, tracing, prioritizing, and controlling changes to requirements throughout the software development lifecycle. However, many problems arise during this process that can negatively affect the project.

1. Changing Requirements (Volatility)

Requirements often change due to evolving business needs, market competition, or new technologies.

If not managed properly, frequent changes cause scope creep (project grows beyond original scope).

👉 **Example:** A client initially asks for a mobile app for payments but later adds crypto integration, increasing complexity and cost.

2. Ambiguous or Incomplete Requirements

Requirements are sometimes unclear, inconsistent, or missing details.

Developers may interpret them differently, leading to mismatched expectations.

👉 **Example:** A requirement says, “The system should be fast” – but “fast” is not defined in measurable terms (2 sec? 5 sec?).

3. Communication Gaps

Poor communication between stakeholders (clients, users, developers, testers) causes misunderstanding.

Different teams may use different terminologies, leading to misinterpretation.

👉 **Example:** The client says “dashboard” but developers design a data table instead of graphs, due to unclear communication.

4. Traceability Issues

Difficulty in linking requirements to design, implementation, and testing.

If traceability is weak, it becomes hard to verify if all requirements are fulfilled.

Created with formatmagic.ai - One click automated formatting

👉 **Example:** A login requirement is mentioned but not linked to test cases, so the feature may be skipped in testing.

5. Prioritization Problems

All requirements are not equally important.

Without proper prioritization, teams may waste time on low-value features while ignoring critical ones.

👉 **Example:** Developers focus on adding a dark mode instead of fixing security requirements.

6. Version Control Issues

Multiple versions of requirements documents may exist.

Without proper versioning, team members may work on outdated requirements.

👉 **Example:** One team uses an old version of the Software Requirement Specification (SRS), causing inconsistency.

7. Conflict Between Stakeholders

Different stakeholders may have conflicting interests.

Balancing these conflicts is challenging for analysts and managers.

👉 **Example:** Marketing wants more features for user attraction, while Finance wants to cut costs.

8. Resource and Time Constraints

Sometimes, the requirements demand more resources (budget, staff, technology) than available.

This leads to project delays or compromises in quality.

👉 **Example:** Client requires advanced AI integration, but the company lacks skilled AI developers.

9. Lack of User Involvement

If users are not actively engaged in requirement gathering and review, the final product may not meet their needs.

Created with formatmagic.ai - One click automated formatting

👉 **Example:** A hospital management system is developed without doctors' input, resulting in irrelevant features.

10. Poor Documentation

If requirements are not properly documented, they may be forgotten, misunderstood, or lost.

Leads to rework, delays, and higher costs.

👉 **Example:** Developers implement a feature differently because the requirement note was informal (verbal) and not in the SRS.

Summary

Requirements management problems mainly arise due to:

- Changing needs (volatility),
- Ambiguity in requirements,
- Communication gaps,
- Traceability and prioritization issues,
- Conflicts, poor documentation, and resource limitations.

These problems, if not addressed, cause delays, cost overruns, poor quality, and customer dissatisfaction.

Created with formatmagic.ai - One click automated formatting

Managing Requirements in an Acquisition Organization

An acquisition organization is a body (often a government agency, large enterprise, or contracting company) that outsources software development to another company or vendor.

Instead of developing the system themselves, they hire external contractors but must still manage the requirements carefully to ensure the delivered product meets their needs.

Key Characteristics of Acquisition Organizations

- They are not developers themselves → They focus on specifying needs.
- Multiple stakeholders are involved → Clients, users, regulators, and contractors.
- Contract-based development → Requirements form the foundation of contracts.
- Emphasis on documentation → Requirements must be precise, measurable, and legally enforceable.

👉 **Example:** A government agency hiring a private software house to build a Tax Management System.

Challenges in Managing Requirements

- Different Perspectives
  - Users describe needs informally.
  - Contractors need precise, technical requirements.
  - Misalignment creates misunderstandings.
- Changes in Contracts
  - Once the contract is signed, changes in requirements may be costly.
- Traceability and Accountability
  - Hard to ensure that each requirement is implemented correctly by contractors.
- Conflict of Interests
  - Acquisition organization wants maximum value at minimum cost,

Created with formatmagic.ai - One click automated formatting

- Contractor wants maximum profit.

Best Practices for Managing Requirements in Acquisition Organizations

1. Clear and Formal Documentation
   - Use Software Requirement Specifications (SRS) and Request for Proposals (RFPs).
   - **Requirements should be:** Complete, Unambiguous, Verifiable, and Measurable.

👉 **Example:** Instead of "system should be fast", write "system should respond within 2 seconds under 1000 concurrent users."

2. Requirements Validation before Contract
   - Validate requirements with stakeholders before finalizing contracts.
   - Use techniques like reviews, prototyping, use cases to confirm accuracy.
3. Change Management Process
   - Define a formal procedure for handling changes.
   - Any change must go through impact analysis (cost, time, resources) before approval.
4. Requirements Traceability
   - Maintain traceability matrices linking requirements → design → implementation → testing.
   - Helps ensure that no requirement is missed and all are testable.
5. Stakeholder Involvement
   - Regular meetings between acquisition org., users, and contractors.
   - Helps bridge the gap between business needs and technical implementation.
6. Monitoring and Control
   - Use periodic reviews, audits, and progress reports to check contractor's compliance with requirements.
   - Employ tools for requirements management (e.g., IBM DOORS, Jama).

Created with formatmagic.ai - One click automated formatting

7. Conflict Resolution Mechanisms

- Include dispute resolution clauses in contracts.
- Balance user satisfaction with project budget/timeline.

Summary

Managing requirements in an acquisition organization is different from internal development because:

- Requirements become legal binding contracts.
- They must be clear, validated, traceable, and monitored.
- Success depends on formal documentation, effective change management, stakeholder involvement, and continuous monitoring.

👉 **In short:**

Good requirement management in acquisition = fewer disputes, controlled costs, satisfied users, and high-quality product delivery.

Created with formatmagic.ai - One click automated formatting

Supplier Organizations

A supplier organization is a company, group, or vendor that is responsible for developing and delivering a software system or service to an acquisition organization (the client).

While the acquisition organization specifies what they need, the supplier organization focuses on how to build and deliver it.

Role of Supplier Organizations

Interpret Client Requirements

- Understand the client's documented needs (e.g., from SRS, RFP, or contract).
- Clarify ambiguities through discussions and prototypes.

Develop the Software System

- Use appropriate software engineering models (Agile, Waterfall, Spiral, etc.).
- Ensure requirements are implemented correctly.

Manage Changing Requirements

- Assess the impact of requirement changes on cost, schedule, and quality.
- Negotiate with the acquisition organization when changes occur.

Deliver High-Quality Products

- Provide a solution that is testable, maintainable, and aligned with requirements.

👉 **Example:** A private IT company (supplier) building an Online Banking System for a national bank (acquisition organization).

Challenges for Supplier Organizations

Ambiguous Requirements

- Client requirements may be incomplete, inconsistent, or unclear.

Created with formatmagic.ai - One click automated formatting

Conflict of Priorities

- Acquisition organization wants high quality at low cost, while suppliers want profit.

Scope Creep

- Frequent requirement changes increase cost and delay delivery.

Traceability Issues

- Hard to link client requirements with actual implementation and testing.

Cultural/Communication Barriers

- If the supplier is offshore, misunderstandings due to language or culture may occur.

Best Practices in Requirement Management for Supplier Organizations

1. Requirement Analysis and Clarification
   - Conduct requirement workshops, interviews, and prototyping with clients.
   - Resolve ambiguities before design starts.
2. Formal Documentation
   - Maintain Software Requirement Specification (SRS).
   - Clearly define acceptance criteria (how the client will verify requirements).
3. Requirements Traceability
   - Maintain a traceability matrix (Requirement → Design → Code → Test).
   - Ensures all client needs are covered and testable.
4. Change Control Process
   - Define a formal change management system.
   - For every new requirement → perform impact analysis → adjust budget & timeline.

Created with formatmagic.ai - One click automated formatting

5. Stakeholder Communication
    - Schedule regular meetings, demos, and progress reports with the acquisition organization.
    - Use tools (like Jira, Confluence, IBM DOORS) to maintain transparency.
6. Quality Assurance and Validation
    - Perform testing (unit, integration, acceptance) to ensure compliance with client needs.
    - Provide documentation and reports as evidence.

Difference Between Acquisition & Supplier Organizations

ASPECT	ACQUISITION ORGANIZATION (CLIENT)	SUPPLIER ORGANIZATION (DEVELOPER)
MAIN ROLE	Defines what is needed	Delivers how it is built
FOCUS	Business needs, user goals	Technical implementation
OUTPUT	Requirement specifications, contracts	Software product, documentation
CHALLENGE	Specifying complete requirements	Interpreting and implementing requirements
CONCERN	Cost, quality, delivery on time	Profitability, efficiency, client satisfaction

Summary

A supplier organization develops software based on requirements provided by the acquisition organization.

Success depends on clear requirement analysis, formal documentation, traceability, change management, communication, and quality assurance.

Effective management helps avoid scope creep, cost overruns, and client dissatisfaction.

Created with formatmagic.ai - One click automated formatting

Product Organizations

A product organization is a type of company that develops software products for a wide range of customers rather than for a single acquisition organization (client).

Unlike supplier organizations that build custom solutions for specific clients, product organizations focus on generic, market-driven software products.

Examples of Product Organizations

- Microsoft (MS Office, Windows)
- Adobe (Photoshop, Illustrator)
- Oracle (Databases)
- Game development studios (EA, Ubisoft)

- 👉 Instead of developing a system for one bank, Microsoft develops MS Excel, which can be used by all banks, businesses, and individuals.

Role of Product Organizations in Requirements Management

Market-Driven Requirements

- Gather requirements by studying market trends, customer feedback, competitors, and technology.
- No single customer defines requirements → they emerge from many stakeholders.

Feature Selection and Prioritization

- Decide which features to include in the product release.
- Prioritize based on market demand, cost, and technical feasibility.

Balancing Stakeholder Needs

- Different customers want different features → organization must balance them.
- **Example:** A business user may want security, while a gamer may want performance.

Created with formatmagic.ai - One click automated formatting

Continuous Updates & Evolution

- Products are never final.
- Regular updates, patches, and new versions are released to keep up with competition and customer needs.

Challenges for Product Organizations

Diverse Customer Base

- Hard to satisfy all types of customers with one product.

Unclear Requirements

- Unlike acquisition projects, requirements are not formally given by one client.

Prioritization Conflict

- Deciding which features to include and which to postpone.

Time-to-Market Pressure

- Must release quickly to beat competitors, even if requirements are incomplete.

Scope Management

- Avoiding "feature bloat" (adding unnecessary features just to attract customers).

Best Practices in Requirement Management for Product Organizations

1. Market Research & Analysis

- Conduct surveys, focus groups, competitor analysis, beta testing.
- **Example:** Adobe releases beta versions of Photoshop to gather feedback before final release.

2. Requirements Prioritization Techniques

- **Use models like:**
  - MoSCoW (Must, Should, Could, Won't)

Created with formatmagic.ai - One click automated formatting

    - Kano Model (Basic, Performance, Excitement features)
- Helps in selecting features that maximize customer satisfaction + profitability.

3. Product Roadmapping

- Maintain a product roadmap showing planned features across upcoming versions.
- **Example:** Windows 10 planned roadmap for regular updates.

4. Incremental Development & Releases

- Use Agile/Iterative models.
- Deliver minimum viable product (MVP) first, then add features gradually.

5. Feedback Loops

- Use customer feedback channels, online reviews, crash reports, analytics tools.
- Helps refine requirements for future releases.

Comparison: Supplier vs Product Organization

ASPECT	SUPPLIER ORGANIZATION	PRODUCT ORGANIZATION
CUSTOMER	Single acquisition org (specific client)	Many customers (broad market)
REQUIREMENTS SOURCE	Given formally by client	Gathered from market research, users, competitors
OUTPUT	Custom solution for one client	Generic product for many
FOCUS	Meeting client contract	Meeting market demand & profitability
CHALLENGE	Client-specific changes	Diverse customer needs & prioritization

Created with formatmagic.ai - One click automated formatting

Summary

- Product organizations develop software for broad markets, not just one client.
- Requirements are market-driven, not contract-driven.
- Main tasks include market research, feature prioritization, product roadmapping, and continuous updates.
- The biggest challenge is satisfying diverse users while staying competitive in the market.

- 👉 **In short:**

> Product organizations must manage requirements strategically to deliver competitive, market-driven products that evolve over time.

Created with formatmagic.ai - One click automated formatting

Requirements Engineering for Agile Methods

1. Introduction

Traditional Software Requirement Engineering (SRE) follows a plan-driven approach: requirements are gathered, documented, and finalized before development begins.

However, Agile methods (like Scrum, XP, Kanban) are adaptive and iterative, where requirements evolve continuously with feedback.

In Agile, requirements engineering is not a one-time phase but an ongoing activity integrated throughout development.

2. Key Features of Agile Requirements Engineering

Incremental Requirement Development

Requirements are broken into small, manageable units (user stories, epics, features).

These are developed in short iterations/sprints.

Example: Instead of fully specifying a shopping cart system upfront, Agile defines user stories like:

- "As a user, I want to add items to my cart so that I can purchase them later."
- "As a user, I want to remove items from my cart so that I only buy what I need."

Customer Collaboration

Agile places the customer at the center of requirement activities.

Requirements evolve based on continuous feedback from stakeholders.

Customers act as Product Owners or on-site representatives.

Flexibility and Change Management

Agile accepts that requirements will change due to market conditions, technology updates, or customer feedback.

Created with formatmagic.ai - One click automated formatting

Instead of rigid contracts, Agile teams embrace change at any stage.

Minimal Documentation

Unlike traditional SRE (which produces detailed requirement documents), Agile uses lightweight documentation.

Requirements are mostly captured in:

- User Stories
- Acceptance Criteria
- Backlogs

3. Agile Requirement Artifacts

Product Backlog

A prioritized list of features, enhancements, and fixes.

Managed by the Product Owner.

Sprint Backlog

Subset of backlog items chosen for development in a sprint.

User Stories

Simple requirement statements written from a user's perspective.

Format: "As a [user], I want [feature] so that [benefit]."

Epics and Features

Larger requirements broken into smaller stories for implementation.

4. Agile Requirement Engineering Techniques

User Story Mapping

Visual organization of stories to understand user journeys.

Backlog Grooming (Refinement)

Continuous updating, prioritizing, and detailing of backlog items.

Created with formatmagic.ai - One click automated formatting

Prototyping & Wireframing

Quickly build mockups to validate requirements before development.

Acceptance Test-Driven Development (ATDD)

Requirements are defined as acceptance tests that guide coding.

5. Benefits of Agile Requirements Engineering

- Faster response to changing needs.
- Strong customer involvement ensures the product matches expectations.
- Reduced risk of building unnecessary features.
- Incremental delivery provides business value earlier.

6. Challenges in Agile Requirements Engineering

- Incomplete Documentation → may cause issues in large projects.
- Customer Availability → requires constant stakeholder involvement.
- Scope Creep → flexible changes may lead to uncontrolled growth.
- Scaling Issues → hard to manage Agile requirements in very large systems.

7. Example Case Study

A company developing a mobile banking app uses Agile:

User stories include:

- “As a customer, I want to check my account balance instantly.”
- “As a customer, I want to transfer funds securely to other accounts.”

Instead of full specifications, developers build features in sprints, get feedback, and refine requirements.

8. Summary (Exam Answer Form)

Requirements Engineering in Agile methods focuses on flexibility, customer collaboration, and incremental development. Unlike traditional approaches that rely on comprehensive documentation, Agile uses user stories, backlogs, and continuous feedback to capture requirements. While this approach improves

Created with formatmagic.ai - One click automated formatting

adaptability and ensures customer satisfaction, it also poses challenges in documentation, scaling, and scope management. Agile RE is best suited for dynamic environments where requirements evolve rapidly.

Created with formatmagic.ai - One click automated formatting

UML (Unified Modeling Language)

1. Introduction

UML is a standardized modeling language used to visualize, specify, construct, and document the artifacts of a software system.

It provides a graphical representation of software design, making it easier to understand, communicate, and manage complex systems.

Developed by Grady Booch, Ivar Jacobson, and James Rumbaugh (the "Three Amigos") in the 1990s.

📌 **Example:** For an online shopping system, UML diagrams can represent user login, order processing, payment workflow, and database structure.

2. Objectives of UML

- **Visualization:** Show system components and their relationships.
- **Specification:** Define system structure and behavior in detail.
- **Construction:** Serve as a blueprint for software development.
- **Documentation:** Provide clear, standardized documentation for future maintenance.

3. Types of UML Diagrams

UML diagrams are divided into two main categories:

A. Structural Diagrams (Static View – What the system is)

- **Class Diagram**

  Shows classes, attributes, methods, and relationships.

  Example: Class User has attributes username, password and methods login(), logout().
- **Object Diagram**

  Snapshot of objects and their relationships at a particular time.
- **Component Diagram**

  Shows software components and their dependencies.

Created with formatmagic.ai - One click automated formatting

Example: Login module depends on Authentication service.

- **Package Diagram**
  Organizes classes into packages for modularity.
- **Deployment Diagram**
  Shows physical nodes (servers, devices) and deployment of components.
- **Composite Structure Diagram**
  Shows internal structure of a class and its interactions.
- **Profile Diagram**
  Extends UML for specific domains (e.g., real-time systems).

B. Behavioral Diagrams (Dynamic View – How the system behaves)

- **Use Case Diagram**
  Represents actors (users) and their interactions with the system.
  Example: Actor Customer → Use cases: Place Order, Make Payment.
- **Sequence Diagram**
  Shows object interactions over time in a sequence.
  Example: Customer sends order → System validates → Payment processed.
- **Activity Diagram**
  Models workflow or business process.
  Example: Steps of order processing from selection → payment → shipment.
- **State Machine Diagram**
  Shows state changes of an object based on events.
  Example: Order states: New → Paid → Shipped → Delivered.
- **Communication Diagram**
  Focuses on object interactions and links, similar to sequence diagram but emphasizes relationships.
- **Interaction Overview Diagram**
  Combines activity and sequence diagrams to show overall interaction.
- **Timing Diagram**
  Shows state changes over time, useful for real-time systems.

4. Key Elements of UML

- **Actors:** Entities interacting with the system (human, external system).
- **Classes:** Blueprint of objects with attributes and methods.
- **Objects:** Instance of a class.

Created with formatmagic.ai - One click automated formatting

- **Relationships:**
  - **Association:** General connection between classes.
  - **Inheritance (Generalization):** Parent-child class relationship.
  - **Aggregation/Composition:** Whole-part relationship.
  - **Dependency:** One element depends on another.
- **Messages:** Communication between objects in behavioral diagrams.

5. Benefits of UML

- Provides standard notation understood by all stakeholders.
- Helps in designing complex systems efficiently.
- Facilitates communication between developers, testers, and clients.
- Supports analysis, design, and documentation of software.
- Allows early detection of errors before implementation.

6. Example – Online Shopping System (UML Overview)

Use Case Diagram:

- **Actors:** Customer, Admin
- **Use Cases:** Browse Products, Add to Cart, Make Payment, Manage Inventory

Class Diagram:

- **Classes:** User, Product, Order, Payment
- **Relationships:** User places Order; Order contains Product; Order uses Payment

Sequence Diagram (Place Order):

- **Customer → System:** Select Product → Add to Cart → Checkout → Payment Confirmation

7. Summary

- UML is a visual modeling language for designing and documenting software systems.
- It consists of structural diagrams (static) and behavioral diagrams (dynamic).
- **Key benefits:** standardization, communication, clarity, and early error detection.

Created with formatmagic.ai - One click automated formatting

- UML is widely used in both traditional and agile software development.

Created with formatmagic.ai - One click automated formatting